

MySQL 5.1 Reference Manual

MySQL 5.1 Reference Manual

Abstract

This is the MySQL Reference Manual. It documents MySQL 5.1 through 5.1.21-beta.

Document generated on: 2007-09-12 (revision: 7802)

Copyright 1997-2007 MySQL AB

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms: You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how MySQL disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of MySQL AB. MySQL AB reserves any and all rights to this documentation not expressly granted above.

Please email [<docs@mysql.com>](mailto:docs@mysql.com) for more information or if you are interested in doing a translation.

Table of Contents

Preface	xxiii
1. General Information	1
1.1. About This Manual	1
1.2. Conventions Used in This Manual	2
1.3. Overview of MySQL AB	3
1.4. Overview of the MySQL Database Management System	4
1.4.1. What is MySQL?	4
1.4.2. History of MySQL	5
1.4.3. The Main Features of MySQL	5
1.5. Overview of the MaxDB Database Management System	8
1.5.1. What is MaxDB?	8
1.5.2. History of MaxDB	8
1.5.3. Features of MaxDB	9
1.5.4. Licensing and Support	9
1.5.5. Feature Differences Between MaxDB and MySQL	9
1.5.6. Interoperability Features Between MaxDB and MySQL	10
1.5.7. MaxDB-Related Links	10
1.6. MySQL Development Roadmap	10
1.6.1. What's New in MySQL 5.1	11
1.6.2. What's Planned for MySQL 5.2	12
1.7. MySQL Information Sources	13
1.7.1. MySQL Mailing Lists	13
1.7.2. MySQL Community Support at the MySQL Forums	15
1.7.3. MySQL Community Support on Internet Relay Chat (IRC)	15
1.7.4. MySQL Enterprise	16
1.8. How to Report Bugs or Problems	16
1.9. MySQL Standards Compliance	19
1.9.1. What Standards MySQL Follows	20
1.9.2. Selecting SQL Modes	20
1.9.3. Running MySQL in ANSI Mode	20
1.9.4. MySQL Extensions to Standard SQL	20
1.9.5. MySQL Differences from Standard SQL	23
1.9.6. How MySQL Deals with Constraints	27
2. Installing and Upgrading MySQL	31
2.1. General Installation Issues	31
2.1.1. Operating Systems Supported by MySQL Community Server	32
2.1.2. Choosing Which MySQL Distribution to Install	33
2.1.3. How to Get MySQL	42
2.1.4. Verifying Package Integrity Using MD5 Checksums or GnuPG	42
2.1.5. Installation Layouts	44
2.2. Standard MySQL Installation Using a Binary Distribution	45
2.3. Installing MySQL on Windows	46
2.3.1. Choosing An Installation Package	46
2.3.2. Installing MySQL with the Automated Installer	47
2.3.3. Using the MySQL Installation Wizard	47
2.3.4. Using the Configuration Wizard	49
2.3.5. Installing MySQL from a Noinstall Zip Archive	53
2.3.6. Extracting the Install Archive	53
2.3.7. Creating an Option File	54
2.3.8. Selecting a MySQL Server Type	55
2.3.9. Starting the Server for the First Time	55
2.3.10. Starting MySQL from the Windows Command Line	56
2.3.11. Starting MySQL as a Windows Service	57
2.3.12. Testing The MySQL Installation	59
2.3.13. Troubleshooting a MySQL Installation Under Windows	59
2.3.14. Upgrading MySQL on Windows	60
2.3.15. MySQL on Windows Compared to MySQL on Unix	61
2.4. Installing MySQL from RPM Packages on Linux	63
2.5. Installing MySQL on Mac OS X	65

2.6. Installing MySQL on Solaris	67
2.7. Installing MySQL on NetWare	68
2.8. Installing MySQL from tar.gz Packages on Other Unix-Like Systems	70
2.9. MySQL Installation Using a Source Distribution	72
2.9.1. Source Installation Overview	73
2.9.2. Typical configure Options	75
2.9.3. Installing from the Development Source Tree	78
2.9.4. Dealing with Problems Compiling MySQL	81
2.9.5. MIT-pthreads Notes	83
2.9.6. Installing MySQL from Source on Windows	84
2.9.7. Compiling MySQL Clients on Windows	87
2.10. Post-Installation Setup and Testing	87
2.10.1. Windows Post-Installation Procedures	87
2.10.2. Unix Post-Installation Procedures	88
2.10.3. Securing the Initial MySQL Accounts	96
2.11. Upgrading MySQL	99
2.11.1. Upgrading from MySQL 5.0 to 5.1	100
2.11.2. Copying MySQL Databases to Another Machine	103
2.12. Downgrading MySQL	104
2.12.1. Downgrading to MySQL 5.0	105
2.13. Operating System-Specific Notes	105
2.13.1. Linux Notes	105
2.13.2. Mac OS X Notes	111
2.13.3. Solaris Notes	111
2.13.4. BSD Notes	115
2.13.5. Other Unix Notes	117
2.13.6. OS/2 Notes	130
2.14. Environment Variables	131
2.15. Perl Installation Notes	131
2.15.1. Installing Perl on Unix	132
2.15.2. Installing ActiveState Perl on Windows	132
2.15.3. Problems Using the Perl DBI/DBD Interface	133
2.16. Porting to Other Systems	135
2.16.1. Debugging a MySQL Server	136
2.16.2. Debugging a MySQL Client	140
2.16.3. The DBUG Package	140
2.16.4. Comments about RTS Threads	141
2.16.5. Differences Between Thread Packages	142
3. Tutorial	144
3.1. Connecting to and Disconnecting from the Server	144
3.2. Entering Queries	145
3.3. Creating and Using a Database	147
3.3.1. Creating and Selecting a Database	148
3.3.2. Creating a Table	148
3.3.3. Loading Data into a Table	150
3.3.4. Retrieving Information from a Table	150
3.4. Getting Information About Databases and Tables	161
3.5. Using mysql in Batch Mode	162
3.6. Examples of Common Queries	163
3.6.1. The Maximum Value for a Column	164
3.6.2. The Row Holding the Maximum of a Certain Column	164
3.6.3. Maximum of Column per Group	164
3.6.4. The Rows Holding the Group-wise Maximum of a Certain Field	165
3.6.5. Using User-Defined Variables	165
3.6.6. Using Foreign Keys	165
3.6.7. Searching on Two Keys	167
3.6.8. Calculating Visits Per Day	167
3.6.9. Using AUTO_INCREMENT	167
3.7. Queries from the Twin Project	169
3.7.1. Find All Non-distributed Twins	169
3.7.2. Show a Table of Twin Pair Status	170
3.8. Using MySQL with Apache	171
4. Using MySQL Programs	172
4.1. Overview of MySQL Programs	172
4.2. Invoking MySQL Programs	173

4.3. Specifying Program Options	173
4.3.1. Using Options on the Command Line	174
4.3.2. Using Option Files	176
4.3.3. Using Options to Set Program Variables	180
4.4. Setting Environment Variables	180
5. Database Administration	182
5.1. Overview of Server-Side Programs	182
5.2. <code>mysqld</code> — The MySQL Server	183
5.2.1. Option and Variable Reference	183
5.2.2. Command Options	201
5.2.3. System Variables	212
5.2.4. Using System Variables	246
5.2.5. Status Variables	254
5.2.6. SQL Modes	269
5.2.7. The Shutdown Process	274
5.2.8. Server-Side Help	275
5.3. MySQL Server Startup Programs	276
5.3.1. <code>mysqld_safe</code> — MySQL Server Startup Script	276
5.3.2. <code>mysql.server</code> — MySQL Server Startup Script	279
5.3.3. <code>mysqld_multi</code> — Manage Multiple MySQL Servers	280
5.4. <code>mysqlmanager</code> — The MySQL Instance Manager	283
5.4.1. MySQL Instance Manager Command Options	284
5.4.2. MySQL Instance Manager Configuration Files	287
5.4.3. Starting the MySQL Server with MySQL Instance Manager	288
5.4.4. Instance Manager User and Password Management	289
5.4.5. MySQL Server Instance Status Monitoring	291
5.4.6. Connecting to MySQL Instance Manager	291
5.4.7. MySQL Instance Manager Commands	291
5.5. Installation-Related Programs	295
5.5.1. <code>comp_err</code> — Compile MySQL Error Message File	295
5.5.2. <code>make_win_bin_dist</code> — Package MySQL Distribution as ZIP Archive	296
5.5.3. <code>mysql_fix_privilege_tables</code> — Upgrade MySQL System Tables	296
5.5.4. <code>mysql_install_db</code> — Initialize MySQL Data Directory	297
5.5.5. <code>mysql_secure_installation</code> — Improve MySQL Installation Security	298
5.5.6. <code>mysql_tzinfo_to_sql</code> — Load the Time Zone Tables	298
5.5.7. <code>mysql_upgrade</code> — Check Tables for MySQL Upgrade	299
5.6. General Security Issues	300
5.6.1. General Security Guidelines	300
5.6.2. Making MySQL Secure Against Attackers	302
5.6.3. Security-Related <code>mysqld</code> Options	303
5.6.4. Security Issues with <code>LOAD DATA LOCAL</code>	305
5.6.5. How to Run MySQL as a Normal User	306
5.7. The MySQL Access Privilege System	306
5.7.1. What the Privilege System Does	306
5.7.2. How the Privilege System Works	306
5.7.3. Privileges Provided by MySQL	310
5.7.4. Connecting to the MySQL Server	313
5.7.5. Access Control, Stage 1: Connection Verification	314
5.7.6. Access Control, Stage 2: Request Verification	317
5.7.7. When Privilege Changes Take Effect	319
5.7.8. Causes of <code>Access denied</code> Errors	319
5.7.9. Password Hashing as of MySQL 4.1	323
5.8. MySQL User Account Management	327
5.8.1. MySQL Usernames and Passwords	327
5.8.2. Adding New User Accounts to MySQL	328
5.8.3. Removing User Accounts from MySQL	331
5.8.4. Limiting Account Resources	331
5.8.5. Assigning Account Passwords	332
5.8.6. Keeping Your Password Secure	333
5.8.7. Using Secure Connections	334
5.9. Backup and Recovery	341
5.9.1. Database Backups	341
5.9.2. Example Backup and Recovery Strategy	342
5.9.3. Point-in-Time Recovery	345
5.9.4. Table Maintenance and Crash Recovery	346

5.10. MySQL Localization and International Usage	355
5.10.1. The Character Set Used for Data and Sorting	355
5.10.2. Setting the Error Message Language	356
5.10.3. Adding a New Character Set	357
5.10.4. The Character Definition Arrays	358
5.10.5. String Collating Support	358
5.10.6. Multi-Byte Character Support	359
5.10.7. Problems With Character Sets	359
5.10.8. MySQL Server Time Zone Support	359
5.10.9. MySQL Server Locale Support	362
5.11. MySQL Server Logs	364
5.11.1. Selecting General Query and Slow Query Log Output Destinations	365
5.11.2. The Error Log	366
5.11.3. The General Query Log	367
5.11.4. The Binary Log	368
5.11.5. The Slow Query Log	375
5.11.6. Server Log Maintenance	376
5.12. Running Multiple MySQL Servers on the Same Machine	377
5.12.1. Running Multiple Servers on Windows	379
5.12.2. Running Multiple Servers on Unix	381
5.12.3. Using Client Programs in a Multiple-Server Environment	382
6. Optimization	384
6.1. Optimization Overview	384
6.1.1. MySQL Design Limitations and Tradeoffs	384
6.1.2. Designing Applications for Portability	384
6.1.3. What We Have Used MySQL For	385
6.1.4. The MySQL Benchmark Suite	386
6.1.5. Using Your Own Benchmarks	386
6.2. Optimizing SELECT and Other Statements	387
6.2.1. Optimizing Queries with EXPLAIN	387
6.2.2. Estimating Query Performance	396
6.2.3. Speed of SELECT Queries	396
6.2.4. WHERE Clause Optimization	397
6.2.5. Range Optimization	398
6.2.6. Index Merge Optimization	401
6.2.7. IS NULL Optimization	403
6.2.8. LEFT JOIN and RIGHT JOIN Optimization	404
6.2.9. Nested Join Optimization	405
6.2.10. Outer Join Simplification	409
6.2.11. ORDER BY Optimization	411
6.2.12. GROUP BY Optimization	413
6.2.13. DISTINCT Optimization	415
6.2.14. Optimizing IN/=ANY Subqueries	415
6.2.15. LIMIT Optimization	419
6.2.16. How to Avoid Table Scans	420
6.2.17. Speed of INSERT Statements	420
6.2.18. Speed of UPDATE Statements	422
6.2.19. Speed of DELETE Statements	422
6.2.20. Other Optimization Tips	422
6.3. Locking Issues	424
6.3.1. Internal Locking Methods	425
6.3.2. Table Locking Issues	427
6.3.3. Concurrent Inserts	428
6.3.4. External Locking	428
6.4. Optimizing Database Structure	429
6.4.1. Design Choices	429
6.4.2. Make Your Data as Small as Possible	429
6.4.3. Column Indexes	430
6.4.4. Multiple-Column Indexes	431
6.4.5. How MySQL Uses Indexes	431
6.4.6. The MyISAM Key Cache	434
6.4.7. MyISAM Index Statistics Collection	437
6.4.8. How MySQL Opens and Closes Tables	439
6.4.9. Drawbacks to Creating Many Tables in the Same Database	440
6.5. Optimizing the MySQL Server	440

6.5.1. System Factors and Startup Parameter Tuning	440
6.5.2. Tuning Server Parameters	441
6.5.3. Controlling Query Optimizer Performance	445
6.5.4. The MySQL Query Cache	445
6.5.5. Examining Thread Information	450
6.5.6. How Compiling and Linking Affects the Speed of MySQL	461
6.5.7. How MySQL Uses Memory	462
6.5.8. How MySQL Uses Internal Temporary Tables	463
6.5.9. How MySQL Uses DNS	464
6.6. Disk Issues	464
6.6.1. Using Symbolic Links	465
7. Client and Utility Programs	468
7.1. Overview of Client and Utility Programs	468
7.2. <code>innochecksum</code> — Offline InnoDB File Checksum Utility	470
7.3. <code>my_print_defaults</code> — Display Options from Option Files	471
7.4. <code>myisam_ftdump</code> — Display Full-Text Index information	471
7.5. <code>myisamchk</code> — MyISAM Table-Maintenance Utility	472
7.5.1. <code>myisamchk</code> General Options	474
7.5.2. <code>myisamchk</code> Check Options	475
7.5.3. <code>myisamchk</code> Repair Options	476
7.5.4. Other <code>myisamchk</code> Options	478
7.5.5. <code>myisamchk</code> Memory Usage	478
7.6. <code>myisamlog</code> — Display MyISAM Log File Contents	479
7.7. <code>myisampack</code> — Generate Compressed, Read-Only MyISAM Tables	480
7.8. <code>mysql</code> — The MySQL Command-Line Tool	485
7.8.1. <code>mysql</code> Options	485
7.8.2. <code>mysql</code> Commands	490
7.8.3. <code>mysql</code> Server-Side Help	493
7.8.4. Executing SQL Statements from a Text File	494
7.8.5. <code>mysql</code> Tips	494
7.9. <code>mysqlaccess</code> — Client for Checking Access Privileges	496
7.10. <code>mysqladmin</code> — Client for Administering a MySQL Server	498
7.11. <code>mysqlbinlog</code> — Utility for Processing Binary Log Files	502
7.12. <code>mysqlcheck</code> — A Table Maintenance and Repair Program	508
7.13. <code>mysqldump</code> — A Database Backup Program	511
7.14. <code>mysqlhotcopy</code> — A Database Backup Program	520
7.15. <code>mysqlimport</code> — A Data Import Program	522
7.16. <code>mysqlshow</code> — Display Database, Table, and Column Information	525
7.17. <code>mysqlslap</code> — Load Emulation Client	527
7.18. <code>mysql_convert_table_format</code> — Convert Tables to Use a Given Storage Engine	529
7.19. <code>mysql_find_rows</code> — Extract SQL Statements from Files	530
7.20. <code>mysql_fix_extensions</code> — Make Table Filename Extensions Lowercase	531
7.21. <code>mysql_setpermission</code> — Interactively Set Permissions in Grant Tables	531
7.22. <code>mysql_tableinfo</code> — Generate Database Metadata	532
7.23. <code>mysql_waitpid</code> — Kill Process and Wait for Its Termination	533
7.24. <code>mysql_zap</code> — Kill Processes That Match a Pattern	534
7.25. <code>perror</code> — Explain Error Codes	534
7.26. <code>replace</code> — A String-Replacement Utility	535
7.27. <code>resolveip</code> — Resolve Hostname to IP Address or Vice Versa	536
7.28. <code>resolve_stack_dump</code> — Resolve Numeric Stack Trace Dump to Symbols	536
8. Language Structure	537
8.1. Literal Values	537
8.1.1. Strings	537
8.1.2. Numbers	539
8.1.3. Hexadecimal Values	539
8.1.4. Boolean Values	540
8.1.5. Bit-Field Values	540
8.1.6. <code>NULL</code> Values	540
8.2. Identifiers	540
8.2.1. Identifier Qualifiers	542
8.2.2. Identifier Case Sensitivity	542
8.2.3. Mapping of Identifiers to Filenames	543
8.2.4. Function Name Parsing and Resolution	544
8.3. Reserved Words	547
8.4. User-Defined Variables	550

8.5. Comment Syntax	552
9. Character Set Support	554
9.1. Character Sets and Collations in General	554
9.2. Character Sets and Collations in MySQL	555
9.3. Specifying Character Sets and Collations	556
9.3.1. Server Character Set and Collation	556
9.3.2. Database Character Set and Collation	556
9.3.3. Table Character Set and Collation	557
9.3.4. Column Character Set and Collation	557
9.3.5. Character String Literal Character Set and Collation	558
9.3.6. National Character Set	559
9.3.7. Examples of Character Set and Collation Assignment	560
9.3.8. Compatibility with Other DBMSs	560
9.4. Connection Character Sets and Collations	561
9.5. Collation Issues	562
9.5.1. Using COLLATE in SQL Statements	563
9.5.2. COLLATE Clause Precedence	563
9.5.3. BINARY Operator	563
9.5.4. Some Special Cases Where the Collation Determination Is Tricky	564
9.5.5. Collations Must Be for the Right Character Set	565
9.5.6. Examples of the Effect of Collation	565
9.6. String Repertoire	566
9.7. Operations Affected by Character Set Support	568
9.7.1. Result Strings	568
9.7.2. CONVERT() and CAST()	569
9.7.3. SHOW Statements and INFORMATION_SCHEMA	569
9.8. Unicode Support	571
9.9. UTF-8 for Metadata	571
9.10. Column Character Set Conversion	572
9.11. Character Sets and Collations That MySQL Supports	573
9.11.1. Unicode Character Sets	574
9.11.2. West European Character Sets	576
9.11.3. Central European Character Sets	577
9.11.4. South European and Middle East Character Sets	578
9.11.5. Baltic Character Sets	579
9.11.6. Cyrillic Character Sets	579
9.11.7. Asian Character Sets	580
10. Data Types	584
10.1. Data Type Overview	584
10.1.1. Overview of Numeric Types	584
10.1.2. Overview of Date and Time Types	587
10.1.3. Overview of String Types	588
10.1.4. Data Type Default Values	590
10.2. Numeric Types	591
10.3. Date and Time Types	593
10.3.1. The DATETIME , DATE , and TIMESTAMP Types	594
10.3.2. The TIME Type	598
10.3.3. The YEAR Type	599
10.3.4. Year 2000 Issues and Date Types	599
10.4. String Types	600
10.4.1. The CHAR and VARCHAR Types	600
10.4.2. The BINARY and VARBINARY Types	601
10.4.3. The BLOB and TEXT Types	602
10.4.4. The ENUM Type	603
10.4.5. The SET Type	604
10.5. Data Type Storage Requirements	606
10.6. Choosing the Right Type for a Column	609
10.7. Using Data Types from Other Database Engines	609
11. Functions and Operators	611
11.1. Operator and Function Reference	611
11.2. Operators	617
11.2.1. Operator Precedence	618
11.2.2. Type Conversion in Expression Evaluation	618
11.2.3. Comparison Functions and Operators	619
11.2.4. Logical Operators	624

11.3. Control Flow Functions	625
11.4. String Functions	627
11.4.1. String Comparison Functions	637
11.4.2. Regular Expressions	639
11.5. Numeric Functions	644
11.5.1. Arithmetic Operators	644
11.5.2. Mathematical Functions	646
11.6. Date and Time Functions	652
11.7. What Calendar Is Used By MySQL?	668
11.8. Full-Text Search Functions	668
11.8.1. Boolean Full-Text Searches	671
11.8.2. Full-Text Searches with Query Expansion	673
11.8.3. Full-Text Stopwords	674
11.8.4. Full-Text Restrictions	676
11.8.5. Fine-Tuning MySQL Full-Text Search	677
11.9. Cast Functions and Operators	678
11.10. XML Functions	680
11.11. Other Functions	685
11.11.1. Bit Functions	686
11.11.2. Encryption and Compression Functions	687
11.11.3. Information Functions	692
11.11.4. Miscellaneous Functions	697
11.12. Functions and Modifiers for Use with GROUP BY Clauses	700
11.12.1. GROUP BY (Aggregate) Functions	700
11.12.2. GROUP BY Modifiers	703
11.12.3. GROUP BY and HAVING with Hidden Fields	705
12. SQL Statement Syntax	707
12.1. Data Definition Statements	707
12.1.1. ALTER DATABASE Syntax	707
12.1.2. ALTER LOGFILE GROUP Syntax	707
12.1.3. ALTER SERVER Syntax	708
12.1.4. ALTER TABLE Syntax	708
12.1.5. ALTER TABLESPACE Syntax	715
12.1.6. CREATE DATABASE Syntax	716
12.1.7. CREATE INDEX Syntax	716
12.1.8. CREATE LOGFILE GROUP Syntax	719
12.1.9. CREATE SERVER Syntax	719
12.1.10. CREATE TABLE Syntax	720
12.1.11. CREATE TABLESPACE Syntax	734
12.1.12. DROP DATABASE Syntax	735
12.1.13. DROP INDEX Syntax	735
12.1.14. DROP LOGFILE GROUP Syntax	736
12.1.15. DROP SERVER Syntax	736
12.1.16. DROP TABLE Syntax	736
12.1.17. DROP TABLESPACE Syntax	737
12.1.18. RENAME DATABASE Syntax	737
12.1.19. RENAME TABLE Syntax	737
12.2. Data Manipulation Statements	738
12.2.1. DELETE Syntax	738
12.2.2. DO Syntax	740
12.2.3. HANDLER Syntax	740
12.2.4. INSERT Syntax	741
12.2.5. LOAD DATA INFILE Syntax	747
12.2.6. REPLACE Syntax	754
12.2.7. SELECT Syntax	755
12.2.8. Subquery Syntax	769
12.2.9. TRUNCATE Syntax	777
12.2.10. UPDATE Syntax	777
12.3. MySQL Utility Statements	779
12.3.1. DESCRIBE Syntax	779
12.3.2. HELP Syntax	780
12.3.3. USE Syntax	781
12.4. MySQL Transactional and Locking Statements	782
12.4.1. START TRANSACTION, COMMIT, and ROLLBACK Syntax	782
12.4.2. Statements That Cannot Be Rolled Back	783

12.4.3. Statements That Cause an Implicit Commit	783
12.4.4. SAVEPOINT and ROLLBACK TO SAVEPOINT Syntax	784
12.4.5. LOCK TABLES and UNLOCK TABLES Syntax	785
12.4.6. SET TRANSACTION Syntax	787
12.4.7. XA Transactions	788
12.5. Database Administration Statements	791
12.5.1. Account Management Statements	791
12.5.2. Table Maintenance Statements	800
12.5.3. SET Syntax	805
12.5.4. SHOW Syntax	810
12.5.5. Other Administrative Statements	832
12.6. Replication Statements	836
12.6.1. SQL Statements for Controlling Master Servers	837
12.6.2. SQL Statements for Controlling Slave Servers	839
12.7. SQL Syntax for Prepared Statements	846
13. Storage Engines	849
13.1. Overview of MySQL Storage Engine Architecture	849
13.1.1. The Common Database Server Layer	850
13.1.2. Pluggable Storage Engine Architecture	850
13.2. Supported Storage Engines	851
13.2.1. Choosing a Storage Engine	852
13.2.2. Comparing Transaction and Non-Transaction Engines	852
13.2.3. Other Storage Engines	853
13.3. Setting the Storage Engine	853
13.4. The MyISAM Storage Engine	854
13.4.1. MyISAM Startup Options	855
13.4.2. Space Needed for Keys	857
13.4.3. MyISAM Table Storage Formats	857
13.4.4. MyISAM Table Problems	859
13.5. The InnoDB Storage Engine	860
13.5.1. InnoDB Overview	860
13.5.2. InnoDB Contact Information	861
13.5.3. InnoDB Configuration	861
13.5.4. InnoDB Startup Options and System Variables	866
13.5.5. Creating the InnoDB Tablespace	872
13.5.6. Creating and Using InnoDB Tables	873
13.5.7. Adding and Removing InnoDB Data and Log Files	882
13.5.8. Backing Up and Recovering an InnoDB Database	883
13.5.9. Moving an InnoDB Database to Another Machine	885
13.5.10. InnoDB Transaction Model and Locking	886
13.5.11. InnoDB Performance Tuning Tips	894
13.5.12. Implementation of Multi-Versioning	899
13.5.13. InnoDB Table and Index Structures	899
13.5.14. InnoDB File Space Management and Disk I/O	901
13.5.15. InnoDB Error Handling	903
13.5.16. Restrictions on InnoDB Tables	908
13.5.17. InnoDB Troubleshooting	910
13.6. The MERGE Storage Engine	911
13.6.1. MERGE Table Problems	913
13.7. The MEMORY (HEAP) Storage Engine	915
13.8. The EXAMPLE Storage Engine	916
13.9. The FEDERATED Storage Engine	917
13.9.1. FEDERATED Storage Engine Overview	917
13.9.2. How to create FEDERATED Tables	918
13.9.3. FEDERATED Storage Engine Notes and Tips	920
13.9.4. FEDERATED Storage Engine Resources	922
13.10. The ARCHIVE Storage Engine	922
13.11. The CSV Storage Engine	923
13.11.1. Repairing and Checking CSV Tables	923
13.11.2. CSV Limitations	924
13.12. The BLACKHOLE Storage Engine	924
14. High Availability, Scalability, and DRBD	927
14.1. Using MySQL with DRBD for High Availability	929
14.1.1. Configuring a MySQL and DRBD Environment	930
15. Replication	938

15.1. Replication Configuration	938
15.1.1. How to Set Up Replication	939
15.1.2. Replication Formats	946
15.1.3. Replication Options and Variables	949
15.1.4. Common Replication Administration Tasks	955
15.2. Replication Solutions	957
15.2.1. Using Replication for Backups	957
15.2.2. Using Replication with different Master and Slave Storage Engines	959
15.2.3. Using Replication for Scale-out	960
15.2.4. Replicating Different Databases to Different Slaves	961
15.2.5. Improving Replication Performance	962
15.2.6. Switching Masters During Failover	963
15.2.7. Setting up Replication using SSL	965
15.3. Replication Notes and Tips	967
15.3.1. Replication Features and Issues	967
15.3.2. Replication Compatibility Between MySQL Versions	975
15.3.3. Upgrading a Replication Setup	975
15.3.4. Replication FAQ	975
15.3.5. Troubleshooting Replication	979
15.3.6. How to Report Replication Bugs or Problems	980
15.4. Replication Implementation	980
15.4.1. Replication Implementation Details	981
15.4.2. Replication Relay and Status Files	982
15.4.3. How Servers Evaluate Replication Rules	984
16. MySQL Cluster	987
16.1. MySQL Cluster Overview	987
16.1.1. MySQL Cluster Core Concepts	988
16.1.2. MySQL Cluster Nodes, Node Groups, Replicas, and Partitions	989
16.2. MySQL Cluster 5.1 Carrier Grade Edition	992
16.2.1. Major Differences Between MySQL 5.1 and MySQL Cluster 5.1 Carrier Grade Edition	993
16.2.2. MySQL Cluster 5.1 Carrier Grade Edition Releases	995
16.3. Simple Multi-Computer How-To	997
16.3.1. Hardware, Software, and Networking	999
16.3.2. Multi-Computer Installation	1000
16.3.3. Multi-Computer Configuration	1003
16.3.4. Initial Startup	1004
16.3.5. Loading Sample Data and Performing Queries	1005
16.3.6. Safe Shutdown and Restart	1008
16.4. MySQL Cluster Configuration	1008
16.4.1. Building MySQL Cluster from Source Code	1008
16.4.2. Installing the Cluster Software	1009
16.4.3. Quick Test Setup of MySQL Cluster	1009
16.4.4. Configuration File	1011
16.4.5. Overview of Cluster Configuration Parameters	1038
16.4.6. Configuring Parameters for Local Checkpoints	1046
16.5. Upgrading and Downgrading MySQL Cluster	1048
16.5.1. Performing a Rolling Restart of the Cluster	1048
16.5.2. Cluster Upgrade and Downgrade Compatibility	1050
16.6. Process Management in MySQL Cluster	1053
16.6.1. MySQL Server Process Usage for MySQL Cluster	1053
16.6.2. ndbd — The Storage Engine Node Process	1054
16.6.3. ndb_mgmd — The Management Server Process	1056
16.6.4. ndb_mgm — The Management Client Process	1056
16.6.5. Command Options for MySQL Cluster Processes	1056
16.7. Management of MySQL Cluster	1060
16.7.1. Summary of MySQL Cluster Start Phases	1060
16.7.2. Commands in the MySQL Cluster Management Client	1062
16.7.3. Event Reports Generated in MySQL Cluster	1064
16.7.4. Single User Mode	1069
16.7.5. Quick Reference: MySQL Cluster SQL Statements	1070
16.8. On-line Backup of MySQL Cluster	1072
16.8.1. Cluster Backup Concepts	1072
16.8.2. Using The Management Client to Create a Backup	1073
16.8.3. ndb_restore — Restore a Cluster Backup	1074
16.8.4. Configuration for Cluster Backup	1077

16.8.5. Backup Troubleshooting	1078
16.9. Cluster Utility Programs	1078
16.9.1. <code>ndb_config</code> — Extract NDB Configuration Information	1080
16.9.2. <code>ndb_cpcd</code> — Automate Testing for NDB Development	1082
16.9.3. <code>ndb_delete_all</code> — Delete All Rows from NDB Table	1082
16.9.4. <code>ndb_desc</code> — Describe NDB Tables	1082
16.9.5. <code>ndb_drop_index</code> — Drop Index from NDB Table	1083
16.9.6. <code>ndb_drop_table</code> — Drop NDB Table	1084
16.9.7. <code>ndb_error_reporter</code> — NDB Error-Reporting Utility	1084
16.9.8. <code>ndb_print_backup_file</code> — Print NDB Backup File Contents	1085
16.9.9. <code>ndb_print_schema_file</code> — Print NDB Schema File Contents	1085
16.9.10. <code>ndb_print_sys_file</code> — Print NDB System File Contents	1085
16.9.11. <code>ndbd_redo_log_reader</code> — Check and Print Content of Cluster Redo Log	1085
16.9.12. <code>ndb_select_all</code> — Print Rows from NDB Table	1086
16.9.13. <code>ndb_select_count</code> — Print Row Counts for NDB Tables	1088
16.9.14. <code>ndb_show_tables</code> — Display List of NDB Tables	1088
16.9.15. <code>ndb_size.pl</code> — NDBCluster Size Requirement Estimator	1089
16.9.16. <code>ndb_waiter</code> — Wait for Cluster to Reach a Given Status	1093
16.10. MySQL Cluster Replication	1094
16.10.1. Abbreviations and Symbols	1095
16.10.2. Assumptions and General Requirements	1096
16.10.3. Known Issues in MySQL Cluster Replication	1096
16.10.4. Cluster Replication Schema and Tables	1100
16.10.5. Preparing the Cluster for Replication	1102
16.10.6. Starting Replication (Single Replication Channel)	1103
16.10.7. Using Two Replication Channels	1105
16.10.8. Implementing Failover with MySQL Cluster	1105
16.10.9. MySQL Cluster Backups With Replication	1106
16.10.10. MySQL Cluster Replication Conflict Resolution	1110
16.11. MySQL Cluster Disk Data Tables	1113
16.11.1. Disk Data Objects	1113
16.11.2. Disk Data Storage Requirements	1115
16.11.3. Disk Data Configuration Parameters	1116
16.12. Using High-Speed Interconnects with MySQL Cluster	1116
16.12.1. Configuring MySQL Cluster to use SCI Sockets	1117
16.12.2. Understanding the Impact of Cluster Interconnects	1120
16.13. Known Limitations of MySQL Cluster	1121
16.13.1. Non-Compliance In SQL Syntax	1121
16.13.2. Limits and Differences from Standard MySQL Limits	1122
16.13.3. Limits Relating to Transaction Handling	1123
16.13.4. Error Handling	1124
16.13.5. Limits Associated with Database Objects	1124
16.13.6. Unsupported Or Missing Features	1124
16.13.7. Limitations Relating to Performance	1125
16.13.8. Issues Exclusive to MySQL Cluster	1125
16.13.9. Limitations Relating to Disk Data Storage	1125
16.13.10. Limitations Relating to Multiple Cluster Nodes	1126
16.13.11. Previous MySQL Cluster Issues Resolved in MySQL 5.1	1126
16.14. MySQL Cluster Development Roadmap	1128
16.14.1. MySQL Cluster Changes in MySQL 5.1	1128
16.15. MySQL Cluster Glossary	1129
17. Partitioning	1132
17.1. Overview of Partitioning in MySQL	1132
17.2. Partition Types	1135
17.2.1. <code>RANGE</code> Partitioning	1136
17.2.2. <code>LIST</code> Partitioning	1138
17.2.3. <code>HASH</code> Partitioning	1139
17.2.4. <code>KEY</code> Partitioning	1141
17.2.5. Subpartitioning	1143
17.2.6. How MySQL Partitioning Handles <code>NULL</code> Values	1145
17.3. Partition Management	1148
17.3.1. Management of <code>RANGE</code> and <code>LIST</code> Partitions	1149
17.3.2. Management of <code>HASH</code> and <code>KEY</code> Partitions	1153
17.3.3. Maintenance of Partitions	1154
17.3.4. Obtaining Information About Partitions	1155

17.4. Partition Pruning	1157
17.5. Restrictions and Limitations on Partitioning	1159
17.5.1. Partitioning Keys, Primary Keys, and Unique Keys	1160
17.5.2. Partitioning Limitations Relating to Storage Engines	1163
17.5.3. Partitioning Limitations Relating to Functions	1163
18. Spatial Extensions	1166
18.1. Introduction to MySQL Spatial Support	1166
18.2. The OpenGIS Geometry Model	1167
18.2.1. The Geometry Class Hierarchy	1167
18.2.2. Class <code>Geometry</code>	1168
18.2.3. Class <code>Point</code>	1169
18.2.4. Class <code>Curve</code>	1169
18.2.5. Class <code>LineString</code>	1169
18.2.6. Class <code>Surface</code>	1170
18.2.7. Class <code>Polygon</code>	1170
18.2.8. Class <code>GeometryCollection</code>	1170
18.2.9. Class <code>MultiPoint</code>	1171
18.2.10. Class <code>MultiCurve</code>	1171
18.2.11. Class <code>MultiLineString</code>	1171
18.2.12. Class <code>MultiSurface</code>	1171
18.2.13. Class <code>MultiPolygon</code>	1172
18.3. Supported Spatial Data Formats	1172
18.3.1. Well-Known Text (WKT) Format	1172
18.3.2. Well-Known Binary (WKB) Format	1173
18.4. Creating a Spatially Enabled MySQL Database	1174
18.4.1. MySQL Spatial Data Types	1174
18.4.2. Creating Spatial Values	1174
18.4.3. Creating Spatial Columns	1177
18.4.4. Populating Spatial Columns	1177
18.4.5. Fetching Spatial Data	1178
18.5. Analyzing Spatial Information	1178
18.5.1. Geometry Format Conversion Functions	1179
18.5.2. <code>Geometry</code> Functions	1179
18.5.3. Functions That Create New Geometries from Existing Ones	1184
18.5.4. Functions for Testing Spatial Relations Between Geometric Objects	1185
18.5.5. Relations on Geometry Minimal Bounding Rectangles (MBRs)	1185
18.5.6. Functions That Test Spatial Relationships Between Geometries	1186
18.6. Optimizing Spatial Analysis	1187
18.6.1. Creating Spatial Indexes	1187
18.6.2. Using a Spatial Index	1188
18.7. MySQL Conformance and Compatibility	1190
19. Stored Procedures and Functions	1191
19.1. Stored Routines and the Grant Tables	1191
19.2. Stored Routine Syntax	1192
19.2.1. <code>CREATE PROCEDURE</code> and <code>CREATE FUNCTION</code> Syntax	1192
19.2.2. <code>ALTER PROCEDURE</code> and <code>ALTER FUNCTION</code> Syntax	1195
19.2.3. <code>DROP PROCEDURE</code> and <code>DROP FUNCTION</code> Syntax	1196
19.2.4. <code>CALL</code> Statement Syntax	1196
19.2.5. <code>BEGIN ... END</code> Compound Statement Syntax	1197
19.2.6. <code>DECLARE</code> Statement Syntax	1197
19.2.7. Variables in Stored Routines	1197
19.2.8. Conditions and Handlers	1198
19.2.9. Cursors	1200
19.2.10. Flow Control Constructs	1201
19.3. Stored Procedures, Functions, Triggers, and <code>LAST_INSERT_ID()</code>	1204
19.4. Binary Logging of Stored Routines and Triggers	1204
20. Triggers	1208
20.1. <code>CREATE TRIGGER</code> Syntax	1208
20.2. <code>DROP TRIGGER</code> Syntax	1211
20.3. Using Triggers	1211
21. Event Scheduler	1214
21.1. Event Scheduler Overview	1214
21.2. Event Scheduler Syntax	1216
21.2.1. <code>ALTER EVENT</code> Syntax	1217
21.2.2. <code>CREATE EVENT</code> Syntax	1218

21.2.3. DROP EVENT Syntax	1222
21.3. Event Metadata	1222
21.4. Event Scheduler Status	1222
21.5. The Event Scheduler and MySQL Privileges	1223
21.6. Event Scheduler Limitations and Restrictions	1225
22. Views	1227
22.1. ALTER VIEW Syntax	1227
22.2. CREATE VIEW Syntax	1227
22.3. DROP VIEW Syntax	1233
23. The INFORMATION_SCHEMA Database	1234
23.1. The INFORMATION_SCHEMA SCHEMATA Table	1235
23.2. The INFORMATION_SCHEMA TABLES Table	1236
23.3. The INFORMATION_SCHEMA COLUMNS Table	1237
23.4. The INFORMATION_SCHEMA STATISTICS Table	1238
23.5. The INFORMATION_SCHEMA USER_PRIVILEGES Table	1238
23.6. The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table	1239
23.7. The INFORMATION_SCHEMA TABLE_PRIVILEGES Table	1239
23.8. The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table	1239
23.9. The INFORMATION_SCHEMA CHARACTER_SETS Table	1240
23.10. The INFORMATION_SCHEMA COLLATIONS Table	1240
23.11. The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table	1241
23.12. The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table	1241
23.13. The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table	1241
23.14. The INFORMATION_SCHEMA ROUTINES Table	1242
23.15. The INFORMATION_SCHEMA VIEWS Table	1243
23.16. The INFORMATION_SCHEMA TRIGGERS Table	1244
23.17. The INFORMATION_SCHEMA PLUGINS Table	1246
23.18. The INFORMATION_SCHEMA ENGINES Table	1246
23.19. The INFORMATION_SCHEMA PARTITIONS Table	1246
23.20. The INFORMATION_SCHEMA EVENTS Table	1249
23.21. The INFORMATION_SCHEMA FILES Table	1252
23.22. The INFORMATION_SCHEMA PROCESSLIST Table	1256
23.23. The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table	1257
23.24. The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables	1258
23.25. The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables	1258
23.26. Other INFORMATION_SCHEMA Tables	1258
23.27. Extensions to SHOW Statements	1258
24. Precision Math	1261
24.1. Types of Numeric Values	1261
24.2. DECIMAL Data Type Changes	1261
24.3. Expression Handling	1262
24.4. Rounding Behavior	1264
24.5. Precision Math Examples	1264
25. APIs and Libraries	1268
25.1. libmysqld, the Embedded MySQL Server Library	1268
25.1.1. Overview of the Embedded MySQL Server Library	1268
25.1.2. Compiling Programs with libmysqld	1268
25.1.3. Restrictions When Using the Embedded MySQL Server	1269
25.1.4. Options with the Embedded Server	1269
25.1.5. Embedded Server Examples	1269
25.1.6. Licensing the Embedded Server	1272
25.2. MySQL C API	1272
25.2.1. C API Data types	1272
25.2.2. C API Function Overview	1276
25.2.3. C API Function Descriptions	1279
25.2.4. C API Prepared Statements	1321
25.2.5. C API Prepared Statement Data types	1322
25.2.6. C API Prepared Statement Function Overview	1327
25.2.7. C API Prepared Statement Function Descriptions	1329
25.2.8. C API Prepared statement problems	1348
25.2.9. C API Handling of Multiple Statement Execution	1348
25.2.10. C API Handling of Date and Time Values	1350
25.2.11. C API Threaded Function Descriptions	1351
25.2.12. C API Embedded Server Function Descriptions	1352
25.2.13. Controlling Automatic Reconnect Behavior	1352

25.2.14. Common Questions and Problems When Using the C API	1353
25.2.15. Building Client Programs	1355
25.2.16. How to Make a Threaded Client	1355
25.3. MySQL PHP API	1357
25.3.1. Common Problems with MySQL and PHP	1357
25.3.2. Enabling Both <code>mysql</code> and <code>mysqli</code> in PHP	1357
25.4. MySQL Perl API	1358
25.5. MySQL C++ API	1358
25.6. MySQL Python API	1359
25.7. MySQL Tcl API	1359
25.8. MySQL Eiffel Wrapper	1359
25.9. MySQL Program Development Utilities	1359
25.9.1. <code>mysql2mysql</code> — Convert mSQL Programs for Use with MySQL	1359
25.9.2. <code>mysql_config</code> — Get Compile Options for Compiling Clients	1359
26. Connectors	1361
26.1. MySQL Connector/ODBC	1361
26.1.1. Introduction to Connector/ODBC	1362
26.1.2. Connector/ODBC Installation	1365
26.1.3. Connector/ODBC Configuration	1384
26.1.4. Connector/ODBC Examples	1401
26.1.5. Connector/ODBC Reference	1425
26.1.6. Connector/ODBC Notes and Tips	1430
26.1.7. Connector/ODBC Support	1439
26.2. MySQL Connector/NET	1440
26.2.1. Connector/NET Versions	1441
26.2.2. Connector/NET Installation	1441
26.2.3. Connector/NET Examples and Usage Guide	1447
26.2.4. Connector/NET Reference	1489
26.2.5. Connector/NET Notes and Tips	1590
26.2.6. Connector/NET Support	1605
26.3. MySQL Visual Studio Plugin	1605
26.3.1. Installing the MySQL Visual Studio Plugin	1605
26.3.2. Creating a connection to the MySQL server	1607
26.3.3. Using the MySQL Visual Studio Plugin	1609
26.3.4. Visual Studio Plugin Support	1615
26.4. MySQL Connector/J	1615
26.4.1. Connector/J Versions	1616
26.4.2. Connector/J Installation	1617
26.4.3. Connector/J Examples	1620
26.4.4. Connector/J (JDBC) Reference	1621
26.4.5. Connector/J Notes and Tips	1650
26.4.6. Connector/J Support	1670
26.5. MySQL Connector/MXJ	1672
26.5.1. Introduction to Connector/MXJ	1672
26.5.2. Connector/MXJ Installation	1673
26.5.3. Connector/MXJ Configuration	1677
26.5.4. Connector/MXJ Reference	1680
26.5.5. Connector/MXJ Notes and Tips	1681
26.5.6. Connector/MXJ Support	1685
26.6. Connector/PHP	1686
27. MySQL Proxy	1687
27.1. MySQL Proxy Supported Platforms	1687
27.2. Installing MySQL Proxy	1687
27.2.1. Installing MySQL Proxy from a binary distribution	1688
27.2.2. Installing MySQL Proxy from a source distribution	1688
27.2.3. Installing MySQL Proxy from the Subversion repository	1688
27.3. MySQL Proxy Command Line Options	1689
27.4. MySQL Proxy Scripting	1690
27.4.1. Proxy Scripting Sequence During Query Injection	1691
27.4.2. Internal Structures	1693
27.4.3. Capturing a connection with <code>connect_server()</code>	1696
27.4.4. Examining the handshake with <code>read_handshake()</code>	1696
27.4.5. Examining the authentication credentials with <code>read_auth()</code>	1697
27.4.6. Accessing authentication information with <code>read_auth_result()</code>	1697
27.4.7. Manipulating Queries with <code>read_query()</code>	1698

27.4.8. Manipulating Results with <code>read_query_result()</code>	1698
27.5. Using MySQL Proxy	1700
27.5.1. Using the Administration Interface	1700
28. Extending MySQL	1702
28.1. MySQL Internals	1702
28.1.1. MySQL Threads	1702
28.1.2. MySQL Test Suite	1702
28.2. The MySQL Plugin Interface	1703
28.2.1. Characteristics of the Plugin Interface	1703
28.2.2. Full-Text Parser Plugins	1704
28.2.3. <code>INSTALL PLUGIN</code> Syntax	1705
28.2.4. <code>UNINSTALL PLUGIN</code> Syntax	1706
28.2.5. Writing Plugins	1706
28.3. Adding New Functions to MySQL	1718
28.3.1. Features of the User-Defined Function Interface	1719
28.3.2. <code>CREATE FUNCTION</code> Syntax	1719
28.3.3. <code>DROP FUNCTION</code> Syntax	1719
28.3.4. Adding a New User-Defined Function	1720
28.3.5. Adding a New Native Function	1728
28.4. Adding New Procedures to MySQL	1729
28.4.1. Procedure Analyse	1729
28.4.2. Writing a Procedure	1729
A. MySQL 5.1 Frequently Asked Questions	1730
A.1. MySQL 5.1 FAQ — General	1730
A.2. MySQL 5.1 FAQ — Storage Engines	1731
A.3. MySQL 5.1 FAQ — Server SQL Mode	1732
A.4. MySQL 5.1 FAQ — Stored Procedures	1733
A.5. MySQL 5.1 FAQ — Triggers	1736
A.6. MySQL 5.1 FAQ — Views	1738
A.7. MySQL 5.0 FAQ — <code>INFORMATION_SCHEMA</code>	1739
A.8. MySQL 5.1 FAQ — Migration	1739
A.9. MySQL 5.1 FAQ — Security	1740
A.10. MySQL 5.1 FAQ — MySQL Cluster	1741
A.11. MySQL 5.1 FAQ — MySQL Chinese, Japanese, and Korean Character Sets	1749
A.12. MySQL 5.1 FAQ — Connectors & APIs	1759
A.13. MySQL 5.1 FAQ — Replication	1759
A.14. MySQL 5.1 FAQ — MySQL, DRBD, and Heartbeat	1759
A.14.1. Distributed Replicated Block Device	1759
A.14.2. Linux Heartbeat	1760
A.14.3. DRBD Architecture	1761
A.14.4. DRBD and MySQL Replication	1761
A.14.5. DRBD and File Systems	1762
A.14.6. DRBD and LVM	1763
A.14.7. DRBD and Virtualization	1763
A.14.8. DRBD and Security	1763
A.14.9. DRBD and System Requirements	1764
A.14.10. DRBD and Support and Consulting	1765
B. Errors, Error Codes, and Common Problems	1767
B.1. Problems and Common Errors	1767
B.1.1. How to Determine What Is Causing a Problem	1767
B.1.2. Common Errors When Using MySQL Programs	1768
B.1.3. Installation-Related Issues	1779
B.1.4. Administration-Related Issues	1780
B.1.5. Query-Related Issues	1786
B.1.6. Optimizer-Related Issues	1791
B.1.7. Table Definition-Related Issues	1791
B.1.8. Known Issues in MySQL	1793
B.2. Server Error Codes and Messages	1795
B.3. Client Error Codes and Messages	1832
C. MySQL Change History	1837
C.1. Changes in release 5.1.x (Development)	1837
C.1.1. Changes in release 5.1.23 (Not yet released)	1837
C.1.2. Changes in release 5.1.22 (Not yet released)	1838
C.1.3. Changes in MySQL 5.1.20 Carrier Grade Edition	1839
C.1.4. Changes in release 5.1.21 (16 August 2007)	1841

C.1.5. Changes in release 5.1.20 (25 June 2007)	1851
C.1.6. Changes in release 5.1.19 (25 May 2007)	1859
C.1.7. Changes in MySQL 5.1.19 Carrier Grade Edition	1862
C.1.8. Changes in release 5.1.18 (08 May 2007)	1865
C.1.9. Changes in MySQL 5.1.18 Carrier Grade Edition	1875
C.1.10. Changes in release 5.1.17 (04 April 2007)	1877
C.1.11. Changes in release 5.1.16 (26 February 2007)	1884
C.1.12. Changes in MySQL 5.1.16 Carrier Grade Edition	1889
C.1.13. Changes in release 5.1.15 (25 January 2007)	1890
C.1.14. Changes in MySQL 5.1.15 Carrier Grade Edition	1897
C.1.15. Changes in release 5.1.14 (05 December 2006)	1911
C.1.16. Changes in MySQL 5.1.14 Carrier Grade Edition	1915
C.1.17. Changes in release 5.1.13 (Not released)	1916
C.1.18. Changes in release 5.1.12 (24 October 2006)	1920
C.1.19. Changes in release 5.1.11 (26 May 2006)	1946
C.1.20. Changes in release 5.1.10 (Not released)	1949
C.1.21. Changes in release 5.1.9 (12 April 2006)	1955
C.1.22. Changes in release 5.1.8 (Not released)	1957
C.1.23. Changes in release 5.1.7 (27 February 2006)	1965
C.1.24. Changes in release 5.1.6 (01 February 2006)	1968
C.1.25. Changes in release 5.1.5 (10 January 2006)	1972
C.1.26. Changes in release 5.1.4 (21 December 2005)	1973
C.1.27. Changes in release 5.1.3 (29 November 2005)	1974
C.1.28. Changes in release 5.1.2 (Not released)	1975
C.1.29. Changes in release 5.1.1 (Not released)	1976
C.2. MySQL Connector/ODBC (MyODBC) Change History	1976
C.2.1. Changes in Connector/ODBC 5.0.12 (Not yet released)	1976
C.2.2. Changes in Connector/ODBC 5.0.11 (31 January 2007)	1976
C.2.3. Changes in Connector/ODBC 5.0.10 (14 December 2006)	1976
C.2.4. Changes in Connector/ODBC 5.0.9 (22 November 2006)	1977
C.2.5. Changes in Connector/ODBC 5.0.8 (17 November 2006)	1977
C.2.6. Changes in Connector/ODBC 5.0.7 (08 November 2006)	1978
C.2.7. Changes in Connector/ODBC 5.0.6 (03 November 2006)	1978
C.2.8. Changes in Connector/ODBC 5.0.5 (17 October 2006)	1979
C.2.9. Changes in Connector/ODBC 5.0.3 (Connector/ODBC 5.0 Alpha 3) (20 June 2006)	1979
C.2.10. Changes in Connector/ODBC 5.0.2 (Never released)	1979
C.2.11. Changes in Connector/ODBC 5.0.1 (Connector/ODBC 5.0 Alpha 2) (05 June 2006)	1979
C.2.12. Changes in Connector/ODBC 3.51.19 (10 August 2007)	1980
C.2.13. Changes in Connector/ODBC 3.51.18 (08 August 2007)	1981
C.2.14. Changes in Connector/ODBC 3.51.17 (14 July 2007)	1982
C.2.15. Changes in Connector/ODBC 3.51.16 (14 June 2007)	1983
C.2.16. Changes in Connector/ODBC 3.51.15 (7 May 2007)	1983
C.2.17. Changes in Connector/ODBC 3.51.14 (08 March 2007)	1984
C.2.18. Changes in Connector/ODBC 3.51.13 (Never released)	1984
C.2.19. Changes in Connector/ODBC 3.51.12	1985
C.2.20. Changes in Connector/ODBC 3.51.11	1985
C.3. Connector/NET Change History	1985
C.3.1. Changes in MySQL Connector/NET Version 5.1.3 (Not yet released)	1985
C.3.2. Changes in MySQL Connector/NET Version 5.1.2 (18 June 2007)	1986
C.3.3. Changes in MySQL Connector/NET Version 5.1.1 (23 May 2007)	1986
C.3.4. Changes in MySQL Connector/NET Version 5.1.0 (01 May 2007)	1987
C.3.5. Changes in MySQL Connector/NET Version 5.0.8 (21 August 2007)	1987
C.3.6. Changes in MySQL Connector/NET Version 5.0.7 (18 May 2007)	1988
C.3.7. Changes in MySQL Connector/NET Version 5.0.6 (22 March 2007)	1988
C.3.8. Changes in MySQL Connector/NET Version 5.0.5 (07 March 2007)	1989
C.3.9. Changes in MySQL Connector/NET Version 5.0.4 (Not released)	1990
C.3.10. Changes in MySQL Connector/NET Version 5.0.3 (05 January 2007)	1990
C.3.11. Changes in MySQL Connector/NET Version 5.0.2 (06 November 2006)	1991
C.3.12. Changes in MySQL Connector/NET Version 5.0.1 (01 October 2006)	1991
C.3.13. Changes in MySQL Connector/NET Version 5.0.0 (08 August 2006)	1992
C.3.14. Changes in MySQL Connector/NET Version 1.0.11 (Not yet released)	1993
C.3.15. Changes in MySQL Connector/NET Version 1.0.10 (24 August 2007)	1993
C.3.16. Changes in MySQL Connector/NET Version 1.0.9 (02 February 2007)	1993
C.3.17. Changes in MySQL Connector/NET Version 1.0.8 (20 October 2006)	1994
C.3.18. Changes in MySQL Connector/NET Version 1.0.7 (21 November 2005)	1995

C.3.19. Changes in MySQL Connector/NET Version 1.0.6 (03 October 2005)	1996
C.3.20. Changes in MySQL Connector/NET Version 1.0.5 (29 August 2005)	1996
C.3.21. Changes in MySQL Connector/NET Version 1.0.4 (20 January 2005)	1997
C.3.22. Changes in MySQL Connector/NET Version 1.0.3-gamma (12 October 2004)	1997
C.3.23. Changes in MySQL Connector/NET Version 1.0.2-gamma (15 November 2004)	1998
C.3.24. Changes in MySQL Connector/NET Version 1.0.1-beta2 (27 October 2004)	1998
C.3.25. Changes in MySQL Connector/NET Version 1.0.0 (01 September 2004)	1999
C.3.26. Changes in MySQL Connector/NET Version 0.9.0 (30 August 2004)	2000
C.3.27. Changes in MySQL Connector/NET Version 0.76	2003
C.3.28. Changes in MySQL Connector/NET Version 0.75	2003
C.3.29. Changes in MySQL Connector/NET Version 0.74	2004
C.3.30. Changes in MySQL Connector/NET Version 0.71	2006
C.3.31. Changes in MySQL Connector/NET Version 0.70	2006
C.3.32. Changes in MySQL Connector/NET Version 0.68	2008
C.3.33. Changes in MySQL Connector/NET Version 0.65	2008
C.3.34. Changes in MySQL Connector/NET Version 0.60	2008
C.3.35. Changes in MySQL Connector/NET Version 0.50	2009
C.4. MySQL Visual Studio Plugin Change History	2009
C.4.1. Changes in MySQL Visual Studio Plugin 1.1.3 (Not yet released)	2009
C.4.2. Changes in MySQL Visual Studio Plugin 1.0.2 (Not yet released)	2009
C.4.3. Changes in MySQL Visual Studio Plugin 1.0.1 (4 October 2006)	2009
C.4.4. Changes in MySQL Visual Studio Plugin 1.0.0 (4 October 2006)	2010
C.5. MySQL Connector/J Change History	2010
C.5.1. Changes in MySQL Connector/J 5.1.x	2010
C.5.2. Changes in MySQL Connector/J 5.0.x	2014
C.5.3. Changes in MySQL Connector/J 3.1.x	2023
C.5.4. Changes in MySQL Connector/J 3.0.x	2038
C.5.5. Changes in MySQL Connector/J 2.0.x	2049
C.5.6. Changes in MySQL Connector/J 1.2b (04 July 1999)	2054
C.5.7. Changes in MySQL Connector/J 1.2.x and lower	2054
C.6. MySQL Connector/MXJ Change History	2058
C.6.1. Changes in MySQL Connector/MXJ 5.0.6 (04 May 2007)	2058
C.6.2. Changes in MySQL Connector/MXJ 5.0.5 (14 March 2007)	2058
C.6.3. Changes in MySQL Connector/MXJ 5.0.4 (28 January 2007)	2059
C.6.4. Changes in MySQL Connector/MXJ 5.0.3 (24 June 2006)	2059
C.6.5. Changes in MySQL Connector/MXJ 5.0.2 (15 June 2006)	2060
C.6.6. Changes in MySQL Connector/MXJ 5.0.1 (Never released)	2061
C.6.7. Changes in MySQL Connector/MXJ 5.0.0 (09 December 2005)	2061
C.7. MySQL Proxy Change History	2061
C.7.1. Changes in MySQL MySQL Proxy 0.6.0 (Not yet released)	2061
C.7.2. Changes in MySQL MySQL Proxy 0.5.1 (30 June 2007)	2062
C.7.3. Changes in MySQL MySQL Proxy 0.5.0 (19 June 2007)	2063
D. Limits and Restrictions	2064
D.1. Restrictions on Stored Routines and Triggers	2064
D.2. Restrictions on Server-Side Cursors	2066
D.3. Restrictions on Subqueries	2066
D.4. Restrictions on Views	2068
D.5. Restrictions on XA Transactions	2070
D.6. Limits in MySQL	2070
D.6.1. Limits of Joins	2070
E. Credits	2071
E.1. Developers at MySQL AB	2071
E.2. Contributors to MySQL	2075
E.3. Documenters and translators	2079
E.4. Libraries used by and included with MySQL	2081
E.5. Packages that support MySQL	2081
E.6. Tools that were used to create MySQL	2082
E.7. Supporters of MySQL	2082
Index	2084

List of Figures

13.1. The MySQL architecture using pluggable storage engines	849
13.2. FEDERATED table structure	917
15.1. Using replication to improve the performance during scaleout	960
15.2. Using replication to replicate separate DBs to multiple hosts	961
15.3. Using an additional replication host to improve performance	962
15.4. Redundancy using replication, initial structure	963
15.5. Redundancy using replication, after master failure	965
A.1. Active-Master MySQL server	1762

List of Tables

26.1. Mapping of MySQL Error Numbers to SQLStates	1639

List of Examples

26.1. Obtaining a connection from the <code>DriverManager</code>	1651
26.2. Using <code>java.sql.Statement</code> to execute a <code>SELECT</code> query	1651
26.3. Stored Procedures	1652
26.4. Using <code>Connection.prepareCall()</code>	1652
26.5. Registering output parameters	1653
26.6. Setting <code>CallableStatement</code> input parameters	1653
26.7. Retrieving results and output parameter values	1654
26.8. Retrieving <code>AUTO_INCREMENT</code> column values using <code>Statement.getGeneratedKeys()</code>	1654
26.9. Retrieving <code>AUTO_INCREMENT</code> column values using <code>SELECT LAST_INSERT_ID()</code>	1655
26.10. Retrieving <code>AUTO_INCREMENT</code> column values in <code>Updatable ResultSets</code>	1656
26.11. Using a connection pool with a J2EE application server	1658
26.12. Example of transaction with retry logic	1668

Preface

This is the Reference Manual for the MySQL Database System, version 5.1, through release 5.1.21-beta. It is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.1 and previous versions. If you are using an earlier release of the MySQL software, please refer to the [MySQL 5.0 Reference Manual](#), which covers the 5.0 series of MySQL software releases, or to [MySQL 3.23, 4.0, 4.1 Reference Manual](#), which covers the 3.23, 4.0, and 4.1 series of MySQL software releases. Differences between minor versions of MySQL 5.1 are noted in the present text with reference to release numbers (5.1.x).

Chapter 1. General Information

The MySQL® software delivers a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. MySQL is a registered trademark of MySQL AB.

The MySQL software is Dual Licensed. Users can choose to use the MySQL software as an Open Source product under the terms of the GNU General Public License (<http://www.fsf.org/licenses/>) or can purchase a standard commercial license from MySQL AB. See <http://www.mysql.com/company/legal/licensing/> for more information on our licensing policies.

The following list describes some sections of particular interest in this manual:

- For a discussion about the capabilities of the MySQL Database Server, see [Section 1.4.3, “The Main Features of MySQL”](#).
- For future plans, see [Section 1.6, “MySQL Development Roadmap”](#).
- For installation instructions, see [Chapter 2, *Installing and Upgrading MySQL*](#). For information about upgrading MySQL, see [Section 2.11, “Upgrading MySQL”](#).
- For a tutorial introduction to the MySQL Database Server, see [Chapter 3, *Tutorial*](#).
- For information about configuring and administering MySQL Server, see [Chapter 5, *Database Administration*](#).
- For information about setting up replication servers, see [Chapter 15, *Replication*](#).
- For answers to a number of questions that are often asked concerning the MySQL Database Server and its capabilities, see [Appendix A, *MySQL 5.1 Frequently Asked Questions*](#).
- For a list of currently known bugs and misfeatures, see [Section B.1.8, “Known Issues in MySQL”](#).
- For a list of all the contributors to this project, see [Appendix E, *Credits*](#).
- For a history of new features and bugfixes, see [Appendix C, *MySQL Change History*](#).
- For tips on porting the MySQL Database Software to new architectures or operating systems, see [MySQL Internals: Porting](#).
- For benchmarking information, see the [sql-bench](#) benchmarking directory in your MySQL distribution.

Important

To report errors (often called “bugs”), please use the instructions at [Section 1.8, “How to Report Bugs or Problems”](#).

If you have found a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to [<security@mysql.com>](mailto:security@mysql.com).

1.1. About This Manual

This is the Reference Manual for the MySQL Database System, version 5.1, through release 5.1.21-beta. It is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.1 and previous versions. If you are using an earlier release of the MySQL software, please refer to the [MySQL 5.0 Reference Manual](#), which covers the 5.0 series of MySQL software releases, or to [MySQL 3.23, 4.0, 4.1 Reference Manual](#), which covers the 3.23, 4.0, and 4.1 series of MySQL software releases. Differences between minor versions of MySQL 5.1 are noted in the present text with reference to release numbers (5.1.x).

Because this manual serves as a reference, it does not provide general instruction on SQL or relational database concepts. It also does not teach you how to use your operating system or command-line interpreter.

The MySQL Database Software is under constant development, and the Reference Manual is updated frequently as well. The most recent version of the manual is available online in searchable form at <http://dev.mysql.com/doc/>. Other formats also are available there, including HTML, PDF, and Windows CHM versions.

The Reference Manual source files are written in DocBook XML format. The HTML version and other formats are produced automatically, primarily using the DocBook XSL stylesheets. For information about DocBook, see <http://docbook.org/>

The DocBook XML sources of this manual are available from <http://dev.mysql.com/tech-resources/sources.html>. You can check out a copy of the documentation repository with this command:

```
svn checkout http://svn.mysql.com/svnpublic/mysql/doc/
```

If you have questions about using MySQL, you can ask them using our mailing lists or forums. See [Section 1.7.1, “MySQL Mailing Lists”](#), and [Section 1.7.2, “MySQL Community Support at the MySQL Forums”](#). If you have suggestions concerning additions or corrections to the manual itself, please send them to the documentation team at <docs@mysql.com>.

This manual was originally written by David Axmark and Michael “Monty” Widenius. It is maintained by the MySQL Documentation Team, consisting of Paul DuBois, Stefan Hinz, Jon Stephens, Martin MC Brown, and Peter Lavin. For the many other contributors, see [Appendix E, Credits](#).

The copyright to this manual is owned by the Swedish company MySQL AB. MySQL® and the MySQL logo are registered trademarks of MySQL AB. Other trademarks and registered trademarks referred to in this manual are the property of their respective owners, and are used for identification purposes only.

1.2. Conventions Used in This Manual

This manual uses certain typographical conventions:

- *Text in this style* is used for SQL statements; database, table, and column names; program listings and source code; and environment variables. Example: “To reload the grant tables, use the `FLUSH PRIVILEGES` statement.”
- *Text in this style* indicates input that you type in examples.
- *Text in this style* indicates the names of executable programs and scripts, examples being `mysql` (the MySQL command line client program) and `mysqld` (the MySQL server executable).
- *Text in this style* is used for variable input for which you should substitute a value of your own choosing.
- Filenames and directory names are written like this: “The global `my.cnf` file is located in the `/etc` directory.”
- Character sequences are written like this: “To specify a wildcard, use the `%` character.”
- *Text in this style* is used for emphasis.
- **Text in this style** is used in table headings and to convey especially strong emphasis.

When commands are shown that are meant to be executed from within a particular program, the prompt shown preceding the command indicates which command to use. For example, `shell>` indicates a command that you execute from your login shell, and `mysql>` indicates a statement that you execute from the `mysql` client program:

```
shell> type a shell command here
mysql> type a mysql statement here
```

The “shell” is your command interpreter. On Unix, this is typically a program such as `sh`, `csh`, or `bash`. On Windows, the equivalent program is `command.com` or `cmd.exe`, typically run in a console window.

When you enter a command or statement shown in an example, do not type the prompt shown in the example.

Database, table, and column names must often be substituted into statements. To indicate that such substitution is necessary, this manual uses `db_name`, `tbl_name`, and `col_name`. For example, you might see a statement like this:

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

This means that if you were to enter a similar statement, you would supply your own database, table, and column names, perhaps like this:

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL keywords are not case sensitive and may be written in any lettercase. This manual uses uppercase.

In syntax descriptions, square brackets (‘[’ and ‘]’) indicate optional words or clauses. For example, in the following statement, **IF EXISTS** is optional:

```
DROP TABLE [IF EXISTS] tbl_name
```

When a syntax element consists of a number of alternatives, the alternatives are separated by vertical bars (‘|’). When one member from a set of choices *may* be chosen, the alternatives are listed within square brackets (‘[’ and ‘]’):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

When one member from a set of choices *must* be chosen, the alternatives are listed within braces (‘{’ and ‘}’):

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

An ellipsis (‘...’) indicates the omission of a section of a statement, typically to provide a shorter version of more complex syntax. For example, **INSERT ... SELECT** is shorthand for the form of **INSERT** statement that is followed by a **SELECT** statement.

An ellipsis can also indicate that the preceding syntax element of a statement may be repeated. In the following example, multiple *reset_option* values may be given, with each of those after the first preceded by commas:

```
RESET reset_option [, reset_option] ...
```

Commands for setting shell variables are shown using Bourne shell syntax. For example, the sequence to set the **CC** environment variable and run the **configure** command looks like this in Bourne shell syntax:

```
shell> CC=gcc ./configure
```

If you are using **csh** or **tcsh**, you must issue commands somewhat differently:

```
shell> setenv CC gcc
shell> ./configure
```

1.3. Overview of MySQL AB

MySQL AB is the company of the MySQL founders and main developers. MySQL AB was originally established in Sweden by David Axmark, Allan Larsson, and Michael “Monty” Widenius.

We are dedicated to developing the MySQL database software and promoting it to new users. MySQL AB owns the copyright to the MySQL source code, the MySQL logo and (registered) trademark, and this manual. See [Section 1.4, “Overview of the MySQL Database Management System”](#).

The MySQL core values show our dedication to MySQL and Open Source.

These core values direct how MySQL AB works with the MySQL server software:

- To be the best and the most widely used database in the world
- To be available and affordable by all
- To be easy to use
- To be continuously improved while remaining fast and safe
- To be fun to use and improve
- To be free from bugs

These are the core values of the company MySQL AB and its employees:

- We subscribe to the Open Source philosophy and support the Open Source community

- We aim to be good citizens
- We prefer partners that share our values and mindset
- We answer email and provide support
- We are a virtual company, networking with others
- We work against software patents

The MySQL Web site (<http://www.mysql.com/>) provides the latest information about MySQL and MySQL AB.

By the way, the “AB” part of the company name is the acronym for the Swedish “aktiebolag,” or “stock company.” It translates to “MySQL, Inc.” In fact, MySQL, Inc. and MySQL GmbH are examples of MySQL AB subsidiaries. They are located in the United States and Germany, respectively.

1.4. Overview of the MySQL Database Management System

1.4.1. What is MySQL?

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by MySQL AB. MySQL AB is a commercial company, founded by the MySQL developers. It is a second generation Open Source company that unites Open Source values and methodology with a successful business model.

The MySQL Web site (<http://www.mysql.com/>) provides the latest information about MySQL software and MySQL AB.

- MySQL is a database management system.

A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

- MySQL is a relational database management system.

A relational database stores data in separate tables rather than putting all the data in one big storeroom. This adds speed and flexibility. The SQL part of “MySQL” stands for “Structured Query Language.” SQL is the most common standardized language used to access databases and is defined by the ANSI/ISO SQL Standard. The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, and “SQL:2003” refers to the current version of the standard. We use the phrase “the SQL standard” to mean the current version of the SQL Standard at any time.

- MySQL software is Open Source.

Open Source means that it is possible for anyone to use and modify the software. Anybody can download the MySQL software from the Internet and use it without paying anything. If you wish, you may study the source code and change it to suit your needs. The MySQL software uses the GPL (GNU General Public License), <http://www.fsf.org/licenses/>, to define what you may and may not do with the software in different situations. If you feel uncomfortable with the GPL or need to embed MySQL code into a commercial application, you can buy a commercially licensed version from us. See the MySQL Licensing Overview for more information (<http://www.mysql.com/company/legal/licensing/>).

- The MySQL Database Server is very fast, reliable, and easy to use.

If that is what you are looking for, you should give it a try. MySQL Server also has a practical set of features developed in close co-operation with our users. You can find a performance comparison of MySQL Server with other database managers on our benchmark page. See [Section 6.1.4, “The MySQL Benchmark Suite”](#).

MySQL Server was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. Although under constant development, MySQL Server today offers a rich and useful set of functions. Its connectivity, speed, and security make MySQL Server highly suited for accessing databases on the Internet.

- MySQL Server works in client/server or embedded systems.

The MySQL Database Software is a client/server system that consists of a multi-threaded SQL server that supports different backends, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs).

We also provide MySQL Server as an embedded multi-threaded library that you can link into your application to get a smaller, faster, easier-to-manage standalone product.

- A large amount of contributed MySQL software is available.

It is very likely that your favorite application or language supports the MySQL Database Server.

The official way to pronounce “MySQL” is “My Ess Que Ell” (not “my sequel”), but we don't mind if you pronounce it as “my sequel” or in some other localized way.

1.4.2. History of MySQL

We started out with the intention of using the [mSQL](#) database system to connect to our tables using our own fast low-level (ISAM) routines. However, after some testing, we came to the conclusion that [mSQL](#) was not fast enough or flexible enough for our needs. This resulted in a new SQL interface to our database but with almost the same API interface as [mSQL](#). This API was designed to allow third-party code that was written for use with [mSQL](#) to be ported easily for use with MySQL.

MySQL is named after co-founder Monty Widenius's daughter, My.

The name of the MySQL Dolphin (our logo) is “Sakila,” which was chosen by the founders of MySQL AB from a huge list of names suggested by users in our “Name the Dolphin” contest. The winning name was submitted by Ambrose Twebaze, an Open Source software developer from Swaziland, Africa. According to Ambrose, the feminine name Sakila has its roots in SiSwati, the local language of Swaziland. Sakila is also the name of a town in Arusha, Tanzania, near Ambrose's country of origin, Uganda.

1.4.3. The Main Features of MySQL

This section describes some of the important characteristics of the MySQL Database Software. See also [Section 1.6, “MySQL Development Roadmap”](#), for more information about current and upcoming features. In most respects, it applies to all versions of MySQL. For information about features as they are introduced into MySQL on a series-specific basis, see the “In a Nutshell” section of the appropriate Manual:

- MySQL 4.0 and 4.1: [MySQL 4.0 in a Nutshell](#), and [MySQL 4.1 in a Nutshell](#).
- MySQL 5.0: [MySQL 5.0 in a Nutshell](#).
- MySQL 5.1: [MySQL 5.1 in a Nutshell](#).

Internals and Portability:

- Written in C and C++.
- Tested with a broad range of different compilers.
- Works on many different platforms. See [Section 2.1.1, “Operating Systems Supported by MySQL Community Server”](#).
- Uses GNU Automake, Autoconf, and Libtool for portability.
- The MySQL Server design is multi-layered with independent modules.
- Fully multi-threaded using kernel threads. It can easily use multiple CPUs if they are available.
- Provides transactional and non-transactional storage engines.
- Uses very fast B-tree disk tables ([MyISAM](#)) with index compression.
- Relatively easy to add other storage engines. This is useful if you want to provide an SQL interface for an in-house database.

- A very fast thread-based memory allocation system.
- Very fast joins using an optimized one-sweep multi-join.
- In-memory hash tables, which are used as temporary tables.
- SQL functions are implemented using a highly optimized class library and should be as fast as possible. Usually there is no memory allocation at all after query initialization.
- The MySQL code is tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool (<http://developer.kde.org/~sewardj/>).
- The server is available as a separate program for use in a client/server networked environment. It is also available as a library that can be embedded (linked) into standalone applications. Such applications can be used in isolation or in environments where no network is available.

Data Types:

- Many data types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long, [FLOAT](#), [DOUBLE](#), [CHAR](#), [VARCHAR](#), [TEXT](#), [BLOB](#), [DATE](#), [TIME](#), [DATETIME](#), [TIMESTAMP](#), [YEAR](#), [SET](#), [ENUM](#), and OpenGIS spatial types. See [Chapter 10, Data Types](#).
- Fixed-length and variable-length records.

Statements and Functions:

- Full operator and function support in the [SELECT](#) list and [WHERE](#) clause of queries. For example:

```
mysql> SELECT CONCAT(first_name, ' ', last_name)
-> FROM citizen
-> WHERE income/dependents > 10000 AND age > 30;
```

- Full support for SQL [GROUP BY](#) and [ORDER BY](#) clauses. Support for group functions ([COUNT\(\)](#), [COUNT\(DISTINCT ...\)](#), [AVG\(\)](#), [STD\(\)](#), [SUM\(\)](#), [MAX\(\)](#), [MIN\(\)](#), and [GROUP_CONCAT\(\)](#)).
- Support for [LEFT OUTER JOIN](#) and [RIGHT OUTER JOIN](#) with both standard SQL and ODBC syntax.
- Support for aliases on tables and columns as required by standard SQL.
- [DELETE](#), [INSERT](#), [REPLACE](#), and [UPDATE](#) return the number of rows that were changed (affected). It is possible to return the number of rows matched instead by setting a flag when connecting to the server.
- The MySQL-specific [SHOW](#) statement can be used to retrieve information about databases, storage engines, tables, and indexes. MySQL 5.0 adds support for the [INFORMATION_SCHEMA](#) database, implemented according to standard SQL.
- The [EXPLAIN](#) statement can be used to determine how the optimizer resolves a query.
- Function names do not clash with table or column names. For example, [ABS](#) is a valid column name. The only restriction is that for a function call, no spaces are allowed between the function name and the '(' that follows it. See [Section 8.3, "Reserved Words"](#).
- You can refer to tables from different databases in the same statement.

Security:

- A privilege and password system that is very flexible and secure, and that allows host-based verification.
- Passwords are secure because all password traffic is encrypted when you connect to a server.

Scalability and Limits:

- Handles large databases. We use MySQL Server with databases that contain 50 million records. We also know of users who use MySQL Server with 60,000 tables and about 5,000,000,000 rows.
- Up to 64 indexes per table are allowed (32 before MySQL 4.1.2). Each index may consist of 1 to 16 columns or parts of columns. The maximum index width is 1000 bytes (767 for [InnoDB](#)); before MySQL 4.1.2, the limit is 500 bytes. An index may use a prefix of a column for [CHAR](#), [VARCHAR](#), [BLOB](#), or [TEXT](#) column types.

Connectivity:

- Clients can connect to MySQL Server using several protocols:
 - Clients can connect using TCP/IP sockets on any platform.
 - On Windows systems in the NT family (NT, 2000, XP, 2003, or Vista), clients can connect using named pipes if the server is started with the `--enable-named-pipe` option. In MySQL 4.1 and higher, Windows servers also support shared-memory connections if started with the `--shared-memory` option. Clients can connect through shared memory by using the `--protocol=memory` option.
 - On Unix systems, clients can connect using Unix domain socket files.
- MySQL client programs can be written in many languages. A client library written in C is available for clients written in C or C++, or for any language that provides C bindings.
- APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available, allowing MySQL clients to be written in many languages. See [Chapter 25, APIs and Libraries](#).
- The Connector/ODBC (MyODBC) interface provides MySQL support for client programs that use ODBC (Open Database Connectivity) connections. For example, you can use MS Access to connect to your MySQL server. Clients can be run on Windows or Unix. MyODBC source is available. All ODBC 2.5 functions are supported, as are many others. See [Chapter 26, Connectors](#).
- The Connector/J interface provides MySQL support for Java client programs that use JDBC connections. Clients can be run on Windows or Unix. Connector/J source is available. See [Chapter 26, Connectors](#).
- MySQL Connector/NET enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. MySQL Connector/NET is a fully managed ADO.NET driver written in 100% pure C#. See [Chapter 26, Connectors](#).

Localization:

- The server can provide error messages to clients in many languages. See [Section 5.10.2, “Setting the Error Message Language”](#).
- Full support for several different character sets, including [latin1](#) (cp1252), [german](#), [big5](#), [ujis](#), and more. For example, the Scandinavian characters ‘å’, ‘ä’ and ‘ö’ are allowed in table and column names. Unicode support is available as of MySQL 4.1.
- All data is saved in the chosen character set.
- Sorting and comparisons are done according to the chosen character set and collation (using [latin1](#) and Swedish collation by default). It is possible to change this when the MySQL server is started. To see an example of very advanced sorting, look at the Czech sorting code. MySQL Server supports many different character sets that can be specified at compile time and runtime.
- As of MySQL 4.1, the server time zone can be changed dynamically, and individual clients can specify their own time zone. See [Section 5.10.8, “MySQL Server Time Zone Support”](#).

MySQL Enterprise

For assistance in getting optimal performance from your MySQL server subscribe to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/>.

Clients and Tools:

- MySQL AB provides several client and utility programs. These include both command-line programs such as [mysqldump](#) and [mysqladmin](#), and graphical programs such as MySQL Administrator and MySQL Query Browser.
- MySQL Server has built-in support for SQL statements to check, optimize, and repair tables. These statements are available from the command line through the [mysqlcheck](#) client. MySQL also includes [myisamchk](#), a very fast command-line utility for performing these operations on [MyISAM](#) tables. See [Chapter 7, Client and Utility Programs](#).
- MySQL programs can be invoked with the `--help` or `-?` option to obtain online assistance.

1.5. Overview of the MaxDB Database Management System

MaxDB is a heavy-duty enterprise database. The database management system is SAP-certified.

MaxDB is the new name of a database management system formerly called SAP DB. In 2003 SAP AG and MySQL AB joined a partnership and re-branded the database system to MaxDB. The development of MaxDB has continued since then as it was done before—through the SAP developer team.

MySQL AB cooperates closely with the MaxDB team at SAP around delivering improvements to the MaxDB product. Joint efforts include development of new native drivers to enable more efficient usage of MaxDB in the Open Source community, and improvement of documentation to expand the MaxDB user base. Interoperability features between MySQL and MaxDB database also are seen as important. For example, the new MaxDB Synchronization Manager supports data synchronization from MaxDB to MySQL.

The MaxDB database management system does not share a common code-base with the MySQL database management system. The MaxDB and MySQL database management systems are independent products provided by MySQL AB.

MySQL AB offers a complete portfolio of Professional Services for MaxDB.

1.5.1. What is MaxDB?

MaxDB is an ANSI SQL-92 (entry level) compliant relational database management system (RDBMS) from SAP AG, that is delivered by MySQL AB as well. MaxDB fulfills the needs for enterprise usage: safety, scalability, high concurrency, and performance. It runs on all major operating systems. Over the years it has proven able to run SAP R/3 and terabytes of data in 24×7 operation.

The database development started in 1977 as a research project at the Technical University of Berlin. In the early 1980s it became a database product that subsequently was owned by Nixdorf, Siemens Nixdorf, Software AG, and today by SAP AG. Along the way, it has been named VDN, Reflex, Supra 2, DDB/4, Entire SQL-DB-Server, and ADABAS D. In 1997, SAP took over the software from Software AG and renamed it to SAP DB. Since October 2000, SAP DB sources additionally were released as Open Source under the GNU General Public License (see [GNU General Public License](#)).

In 2003, SAP AG and MySQL AB formed a partnership and re-branded the database system to MaxDB.

1.5.2. History of MaxDB

The history of MaxDB goes back to SAP DB, SAP AG's DBMS. That is, MaxDB is a re-branded and enhanced version of SAP DB. For many years, MaxDB has been used for small, medium, and large installations of the mySAP Business Suite and other demanding SQL applications requiring an enterprise-class DBMS with regard to the number of users, the transactional workload, and the size of the database.

SAP DB was meant to provide an alternative to third-party database systems such as Oracle, Microsoft SQL Server, and DB2 by IBM. In October 2000, SAP AG released SAP DB under the GNU GPL license (see [GNU General Public License](#)), thus making it Open Source software.

Today, MaxDB is used in about 3,500 SAP customer installations worldwide. Moreover, the majority of all DBMS installations on Unix and Linux within SAP's IT department rely on MaxDB. MaxDB is tuned toward heavy-duty online transaction processing (OLTP) with several thousand users and database sizes ranging from several hundred GB to multiple TB.

In 2003, SAP and MySQL concluded a partnership and development cooperation agreement. As a result, SAP's database system SAP DB has been delivered under the name of MaxDB by MySQL since the release of version 7.5 (November 2003).

Version 7.5 of MaxDB is a direct advancement of the SAP DB 7.4 code base. Therefore, the MaxDB software version 7.5 can be used as a direct upgrade of previous SAP DB versions starting 7.2.04 and higher.

The former SAP DB development team at SAP AG is responsible, now as before, for developing and supporting MaxDB. MySQL AB

cooperates closely with the MaxDB team at SAP around delivering improvements to the MaxDB product, see [Section 1.5, “Overview of the MaxDB Database Management System”](#). Both SAP AG and MySQL AB handle the sale and distribution of MaxDB. The advancement of MaxDB and the MySQL Server leverages synergies that benefit both product lines.

MaxDB is subjected to SAP AG's complete quality assurance process before it is shipped with SAP solutions or provided as a download from the MySQL site.

1.5.3. Features of MaxDB

MaxDB is a heavy-duty, SAP-certified Open Source database for OLTP and OLAP usage which offers high reliability, availability, scalability, and a very comprehensive feature set. It is targeted for large mySAP Business Suite environments and other applications that require maximum enterprise-level database functionality and complements the MySQL database server.

MaxDB operates as a client/server product. It was developed to meet the needs of installations in OLTP and Data Warehouse/OLAP/Decision Support scenarios and offers these benefits:

- **Easy configuration and administration:** GUI-based Installation Manager and Database Manager as single administration tools for DBMS operations
- **Around-the-clock operation, no planned downtimes, no permanent attendance required:** Automatic space management, no need for reorganizations
- **Sophisticated backup and restore capabilities:** Online and incremental backups, recovery wizard to guide you through the recovery scenario
- **Supports large number of users, database sizes in the terabytes, and demanding workloads:** Proven reliability, performance, and scalability
- **High availability:** Cluster support, standby configuration, hot standby configuration

1.5.4. Licensing and Support

MaxDB can be used under the same licenses available for the other products distributed by MySQL AB. Thus, MaxDB is available under the GNU General Public License, and a commercial license. For more information on licensing, see <http://www.mysql.com/company/legal/licensing/>.

MySQL AB offers MaxDB technical support to non-SAP customers. MaxDB support is available on various levels (Basic, Silver, and Gold), which expand from unlimited email/web-support to 24×7 phone support for business critical systems.

MySQL AB also offers Licenses and Support for MaxDB when used with SAP Applications, like SAP NetWeaver and mySAP Business Suite. For more information on licenses and support for your needs, please contact MySQL AB. (See <http://www.mysql.com/company/contact/>.)

Consulting and training services are available. MySQL gives classes on MaxDB at regular intervals. See <http://www.mysql.com/training/> for a list of classes.

1.5.5. Feature Differences Between MaxDB and MySQL

MaxDB is MySQL AB's SAP-certified database. The MaxDB database server complements the MySQL AB product portfolio. Some MaxDB features are not available on the MySQL database management server and vice versa.

The following list summarizes the main differences between MaxDB and MySQL; it is not complete.

- MaxDB runs as a client/server system. MySQL can run as a client/server system or as an embedded system.
- MaxDB might not run on all platforms supported by MySQL.
- MaxDB uses a proprietary network protocol for client/server communication. MySQL uses either TCP/IP (with or without SSL encryption), sockets (under Unix-like systems), or named pipes or shared memory (under Windows NT-family systems).
- MaxDB supports stored procedures and functions. MySQL 5.0 and up also supports stored procedures and functions. MaxDB sup-

ports programming of triggers through an SQL extension. MySQL 5.0 supports triggers. MaxDB contains a debugger for stored procedure languages, can cascade nested triggers, and supports multiple triggers per action and row.

- MaxDB is distributed with user interfaces that are text-based, graphical, or Web-based. MySQL is distributed with text-based user interfaces only; graphical user interfaces such as MySQL Query Browser or MySQL Administrator are shipped separately from the main distributions. Web-based user interfaces for MySQL are offered by third parties.
- MaxDB supports a number of programming interfaces that also are supported by MySQL. For developing with MaxDB, the MaxDB ODBC Driver, SQL Database Connectivity (SQLDBC), JDBC Driver, Perl and Python modules and a MaxDB PHP extension, which provides access to MySQL MaxDB databases using PHP, are available. Third Party Programming Interfaces: Support for OLE DB, ADO, DAO, RDO and .NET through ODBC. MaxDB supports embedded SQL with C/C++.
- MaxDB includes administrative features that MySQL does not have: job scheduling by time (included in MySQL as of 5.1), event, and alert, and sending messages to a database administrator on alert thresholds. (MySQL has scheduling support starting with version 5.1.6.)

1.5.6. Interoperability Features Between MaxDB and MySQL

MaxDB and MySQL are independent database management servers. The interoperation of the systems is possible in a way that the systems can exchange their data. To exchange data between MaxDB and MySQL, you can use the import and export tools of the systems or the MaxDB Synchronization Manager. The import and export tools can be used to transfer data in an infrequent, manual fashion. The MaxDB Synchronization Manager offers faster, automatic data transfer capabilities.

The MaxDB Loader can be used to export data and object definitions. The Loader can export data using MaxDB internal, binary formats and text formats (CSV). Data exported from MaxDB in text formats can be imported into MySQL using the `mysqlimport` client program. To export MySQL data, you can use either `mysqldump` to create `INSERT` statements or `SELECT ... INTO OUT-FILE` to create a text file (CSV). Use the MaxDB Loader to import the data files generated by MySQL.

Object definitions can be exchanged between the systems using MaxDB Loader and the MySQL tool `mysqldump`. As the SQL dialects of both systems differ slightly and MaxDB has features currently not supported by MySQL like SQL constraints, we recommend to hand-tune the definition files. The `mysqldump` tool offers an option `--compatible=maxdb` to produce output that is compatible with MaxDB to make porting easier.

The MaxDB Synchronization Manager is available as part of MaxDB 7.6. The Synchronization Manager supports creation of asynchronous replication scenarios between several MaxDB instances. However, interoperability features also are planned, so that the Synchronization Manager supports replication to and from a MySQL server.

1.5.7. MaxDB-Related Links

The main page for MaxDB information is <http://www.mysql.com/products/maxdb>, which provides details about the features of the MaxDB database management systems and has pointers to available documentation.

The MySQL Reference Manual does not contain any MaxDB documentation other than the introduction given in this section. MaxDB has its own documentation, which is called the MaxDB library and is available at <http://dev.mysql.com/doc/maxdb/index.html>.

MySQL AB runs a community mailing list on MaxDB; see <http://lists.mysql.com/maxdb>. The list shows a vivid community discussion. Many of the core developers contribute to it. Product announcements are sent to the list.

A Web forum on MaxDB is available at <http://forums.mysql.com/>. The forum focuses on MaxDB questions not related to SAP applications.

1.6. MySQL Development Roadmap

This section describes the general MySQL development roadmap, provides an overview about features that have been implemented in previous series and that are new in this current release series (5.1), and an overview about upcoming additions or changes in the next release series (5.2).

The maturity level of the release series covered in this manual (5.1) is beta. Information about maturity levels can be found in [Section 2.1.2.1, “Choosing Which Version of MySQL to Install”](#).

Before upgrading from one release series to the next, please see the notes in [Section 2.11, “Upgrading MySQL”](#).

The most requested features and the versions in which they were implemented or are scheduled for implementation are summarized in

the following table:

Feature	MySQL Series
Unions	4.0
Subqueries	4.1
R-trees	4.1 (for the MyISAM storage engine)
Stored procedures	5.0
Views	5.0
Cursors	5.0
XA transactions	5.0
Triggers	5.0 and 5.1
Event scheduler	5.1
Partitioning	5.1
Pluggable storage engine API	5.1
Plugin API	5.1
Row-based replication	5.1
Server log tables	5.1
Foreign keys	5.2 (implemented in 3.23 for InnoDB)

1.6.1. What's New in MySQL 5.1

The following features have been added to MySQL 5.1.

- Partitioning:** This capability enables distributing portions of individual tables across a filesystem, according to rules which can be set when the table is created. In effect, different portions of a table are stored as separate tables in different locations, but from the user point of view, the partitioned table is still a single table. Syntactically, this implements a number of new extensions to the [CREATE TABLE](#), [ALTER TABLE](#), and [EXPLAIN . . . SELECT](#) statements. As of MySQL 5.1.6, queries against partitioned tables can take advantage of *partition pruning*. In some cases, this can result in query execution that is an order of magnitude faster than the same query against a non-partitioned version of the same table. See [Chapter 17, Partitioning](#), for further information on this functionality. (Author: Mikael Ronström)
- Row-based replication:** Replication capabilities in MySQL originally were based on propagation of SQL statements from master to slave. This is called *statement-based replication*. As of MySQL 5.1.5, another basis for replication is available. This is called *row-based replication*. Instead of sending SQL statements to the slave, the master writes events to its binary log that indicate how individual table rows are effected. As of MySQL 5.1.8, a third option is available: *mixed*. This will use statement-based replication by default, and only switch to row-based replication in particular cases. See [Section 15.1.2, “Replication Formats”](#). (Authors: Lars Thalmann, Guilhem Bichot, Mats Kindahl)
- Plugin API:** MySQL 5.1 adds support for a very flexible plugin API that enables loading and unloading of various components at runtime, without restarting the server. Although the work on this is not finished yet, *plugin full-text parsers* are a first step in this direction. This allows users to implement their own input filter on the indexed text, enabling full-text search capability on arbitrary data such as PDF files or other document formats. A pre-parser full-text plugin performs the actual parsing and extraction of the text and hands it over to the built-in MySQL full-text search. See [Section 28.2, “The MySQL Plugin Interface”](#). (Author: Sergey Votovich)
- Event scheduler:** MySQL Events are tasks that run according to a schedule. When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time. Conceptually, this is similar to the idea of the Unix [crontab](#) (also known as a “cron job”) or the Windows Task Scheduler. See [Chapter 21, Event Scheduler](#). (Author: Andrey Hristov)
- Server log tables:** Before MySQL 5.1, the server writes general query log and slow query log entries to log files. As of MySQL 5.1, the server's logging capabilities for these logs are more flexible. Log entries can be written to log files (as before) or to the [general_log](#) and [slow_log](#) tables in the [mysql](#) database. If logging is enabled, either or both destinations can be selected. The [-log-output](#) option controls the destination or destinations of log output. See [Section 5.11.1, “Selecting General Query and Slow Query Log Output Destinations”](#). (Author: Petr Chardin)

- **The Instance Manager (IM)** now has some additional functionality: `SHOW instance_name LOG FILES` provides a listing of all log files, `SHOW instance_name LOG {ERROR | SLOW | GENERAL} size` retrieves a part of the specified log file, and `SET instance_name.option_name=option_value` sets an option to the specified value and writes it to the configuration file. See [Section 5.4, “mysqlmanager — The MySQL Instance Manager”](#). (Author: Petr Chardin)
- **Upgrade program:** The `mysql_upgrade` program (available as of MySQL 5.1.7) checks all existing tables for incompatibilities with the current version of MySQL Server and repairs them if necessary. This program should be run for each MySQL upgrade. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#). (Authors: Alexey Botchkov, Mikael Widén)
- **Replication between MySQL Clusters** is now supported. It is now also possible to replicate between a MySQL Cluster and a non-cluster database. See [Section 16.10, “MySQL Cluster Replication”](#).
- **MySQL Cluster Disk Data:** In MySQL versions previous to 5.1.6, the `NDBCluster` storage engine was strictly in-memory; beginning with MySQL 5.1.6, it is possible to store Cluster data (but not indexes) on disk. This allows MySQL Cluster to scale upward with fewer hardware (RAM) requirements than previously. See [Section 16.11, “MySQL Cluster Disk Data Tables”](#).

The Disk Data implementation includes a new “no-steal” restoration algorithm for fast node restarts when storing very large amounts of data (terabyte range).

- **Online `ADD INDEX` and `DROP INDEX` for MySQL Cluster:** Adding and dropping indexes is now much faster for tables using the `NDB` storage engine than in previous versions of MySQL Cluster. This is due to the fact that `NDB` no longer needs to re-create tables, but instead can now apply these schema changes directly to existing tables.
- **Improved backup implementation in MySQL Cluster:** A fault arising in a single data node during a Cluster backup no longer causes the entire backup to be aborted, as occurred in previous versions of MySQL Cluster.
- **Backup of tablespaces:** The `mysqldump` utility now supports an option for dumping tablespaces. Use `-Y` or `-all-tablespaces` to enable this functionality.
- **Improvements to `INFORMATION_SCHEMA`:** MySQL 5.1 provides much more information in its metadata database. New tables in that database include `FILES`, `EVENTS`, `PARTITIONS`, `PROCESSLIST`, `ENGINES`, and `PLUGINS`.
- **XML functions:** `ExtractValue()` returns the content of a fragment of XML matching a given XPath expression. `UpdateXML()` replaces the element selected from a fragment of XML by an XPath expression supplied by the user with a second XML fragment (also user-supplied), and returns the modified XML. See [Section 11.10, “XML Functions”](#). (Author: Alexander Barkov)
- **Load emulator:** The `mysqlslap` program is designed to emulate client load for a MySQL server and report the timing of each stage. It works as if multiple clients were accessing the server. See [Section 7.17, “mysqlslap — Load Emulation Client”](#). (Authors: Patrick Galbraith, Brian Aker)

1.6.2. What's Planned for MySQL 5.2

This section is subject to change as long as MySQL 5.2 development is in its early stages.

The following features will be added to MySQL 5.2, or change in MySQL 5.2.

- `RESET SLAVE` no longer changes replication connection parameters. Previously, it reset them to the values specified on the command line.

The following constructs are deprecated and have been removed in MySQL 5.2. Where alternatives are shown, applications should be updated to use them.

- The `table_type` system variable (use `storage_engine`)
- The `log_bin_trust_routine_creators` variable (use `log_bin_trust_function_creators`)
- `TIMESTAMP(N)`: The ability to specify a display width of `N` (use without `N`)
- The `TYPE` table option to specify the storage engine for `CREATE TABLE` or `ALTER TABLE` (use `ENGINE`)
- The `SHOW TABLE TYPES` SQL statement (use `SHOW ENGINES`)

- The `SHOW INNODB STATUS` SQL statement (use `SHOW ENGINE INNODB STATUS`)
- The `SHOW MUTEX STATUS` SQL statement (use `SHOW ENGINE INNODB MUTEX`)
- The `LOAD TABLE ... FROM MASTER` and `LOAD DATA FROM MASTER` SQL statements
- The `SHOW PLUGIN` SQL statement (use `SHOW PLUGINS`)
- The `RESTORE TABLE` SQL statement
- The `BACKUP TABLE` SQL statement
- The `--master-xxx` server options to set replication parameters (use the `CHANGE MASTER` statement)

1.7. MySQL Information Sources

This section lists sources of additional information that you may find helpful, such as the MySQL mailing lists and user forums, and Internet Relay Chat.

1.7.1. MySQL Mailing Lists

This section introduces the MySQL mailing lists and provides guidelines as to how the lists should be used. When you subscribe to a mailing list, you receive all postings to the list as email messages. You can also send your own questions and answers to the list.

To subscribe to or unsubscribe from any of the mailing lists described in this section, visit <http://lists.mysql.com/>. For most of them, you can select the regular version of the list where you get individual messages, or a digest version where you get one large message per day.

Please *do not* send messages about subscribing or unsubscribing to any of the mailing lists, because such messages are distributed automatically to thousands of other users.

Your local site may have many subscribers to a MySQL mailing list. If so, the site may have a local mailing list, so that messages sent from lists.mysql.com to your site are propagated to the local list. In such cases, please contact your system administrator to be added to or dropped from the local MySQL list.

If you wish to have traffic for a mailing list go to a separate mailbox in your mail program, set up a filter based on the message headers. You can use either the `List-ID:` or `Delivered-To:` headers to identify list messages.

The MySQL mailing lists are as follows:

- [announce](#)

This list is for announcements of new versions of MySQL and related programs. This is a low-volume list to which all MySQL users should subscribe.

- [mysql](#)

This is the main list for general MySQL discussion. Please note that some topics are better discussed on the more-specialized lists. If you post to the wrong list, you may not get an answer.

- [bugs](#)

This list is for people who want to stay informed about issues reported since the last release of MySQL or who want to be actively involved in the process of bug hunting and fixing. See [Section 1.8, “How to Report Bugs or Problems”](#).

- [internals](#)

This list is for people who work on the MySQL code. This is also the forum for discussions on MySQL development and for posting patches.

- [mysqldoc](#)

This list is for people who work on the MySQL documentation: people from MySQL AB, translators, and other community mem-

bers.

- [benchmarks](#)

This list is for anyone interested in performance issues. Discussions concentrate on database performance (not limited to MySQL), but also include broader categories such as performance of the kernel, filesystem, disk system, and so on.

- [packagers](#)

This list is for discussions on packaging and distributing MySQL. This is the forum used by distribution maintainers to exchange ideas on packaging MySQL and on ensuring that MySQL looks and feels as similar as possible on all supported platforms and operating systems.

- [java](#)

This list is for discussions about the MySQL server and Java. It is mostly used to discuss JDBC drivers such as MySQL Connector/J.

- [win32](#)

This list is for all topics concerning the MySQL software on Microsoft operating systems, such as Windows 9x, Me, NT, 2000, XP, and 2003.

- [myodbc](#)

This list is for all topics concerning connecting to the MySQL server with ODBC.

- [gui-tools](#)

This list is for all topics concerning MySQL graphical user interface tools such as [MySQL Administrator](#) and [MySQL Query Browser](#).

- [cluster](#)

This list is for discussion of MySQL Cluster.

- [dotnet](#)

This list is for discussion of the MySQL server and the .NET platform. It is mostly related to MySQL Connector/Net.

- [plusplus](#)

This list is for all topics concerning programming with the C++ API for MySQL.

- [perl](#)

This list is for all topics concerning Perl support for MySQL with [DBD: :mysql](#).

If you're unable to get an answer to your questions from a MySQL mailing list or forum, one option is to purchase support from MySQL AB. This puts you in direct contact with MySQL developers.

The following table shows some MySQL mailing lists in languages other than English. These lists are not operated by MySQL AB.

- [<mysql-france-subscribe@yahoogroups.com>](mailto:mysql-france-subscribe@yahoogroups.com)

A French mailing list.

- [<list@tinc.net>](mailto:list@tinc.net)

A Korean mailing list. To subscribe, email [subscribe mysql your@email.address](#) to this list.

- [<mysql-de-request@lists.4t2.com>](mailto:mysql-de-request@lists.4t2.com)

A German mailing list. To subscribe, email [subscribe mysql-de your@email.address](#) to this list. You can find inform-

ation about this mailing list at <http://www.4t2.com/mysql/>.

- `<mysql-br-request@listas.linkway.com.br>`

A Portuguese mailing list. To subscribe, email `subscribe mysql-br your@email.address` to this list.

- `<mysql-alta@elistas.net>`

A Spanish mailing list. To subscribe, email `subscribe mysql your@email.address` to this list.

1.7.1.1. Guidelines for Using the Mailing Lists

Please don't post mail messages from your browser with HTML mode turned on. Many users don't read mail with a browser.

When you answer a question sent to a mailing list, if you consider your answer to have broad interest, you may want to post it to the list instead of replying directly to the individual who asked. Try to make your answer general enough that people other than the original poster may benefit from it. When you post to the list, please make sure that your answer is not a duplication of a previous answer.

Try to summarize the essential part of the question in your reply. Don't feel obliged to quote the entire original message.

When answers are sent to you individually and not to the mailing list, it is considered good etiquette to summarize the answers and send the summary to the mailing list so that others may have the benefit of responses you received that helped you solve your problem.

1.7.2. MySQL Community Support at the MySQL Forums

The forums at <http://forums.mysql.com> are an important community resource. Many forums are available, grouped into these general categories:

- Migration
- MySQL Usage
- MySQL Connectors
- Programming Languages
- Tools
- 3rd-Party Applications
- Storage Engines
- MySQL Technology
- SQL Standards
- Business

1.7.3. MySQL Community Support on Internet Relay Chat (IRC)

In addition to the various MySQL mailing lists and forums, you can find experienced community people on Internet Relay Chat (IRC). These are the best networks/channels currently known to us:

freenode (see <http://www.freenode.net/> for servers)

- `#mysql` is primarily for MySQL questions, but other database and general SQL questions are welcome. Questions about PHP, Perl, or C in combination with MySQL are also common.

If you are looking for IRC client software to connect to an IRC network, take a look at **xChat** (<http://www.xchat.org/>). X-Chat (GPL licensed) is available for Unix as well as for Windows platforms (a free Windows build of X-Chat is available at [ht-](#)

[tp://www.silverex.org/download/](http://www.silverex.org/download/)).

1.7.4. MySQL Enterprise

MySQL AB offers technical support in the form of MySQL Enterprise. For organizations that rely on the MySQL DBMS for business-critical production applications, MySQL Enterprise is a commercial subscription offering which includes:

- MySQL Enterprise Server
- MySQL Enterprise Monitor
- Monthly Rapid Updates and Quarterly Service Packs
- MySQL Knowledge Base
- 24x7 Technical and Consultative Support

MySQL Enterprise is available in multiple tiers, giving you the flexibility to choose the level of service that best matches your needs. For more information see [MySQL Enterprise](#).

1.8. How to Report Bugs or Problems

Before posting a bug report about a problem, please try to verify that it is a bug and that it has not been reported already:

- Start by searching the MySQL online manual at <http://dev.mysql.com/doc/>. We try to keep the manual up to date by updating it frequently with solutions to newly found problems. The change history (<http://dev.mysql.com/doc/mysql/en/news.html>) can be particularly useful since it is quite possible that a newer version contains a solution to your problem.
- If you get a parse error for a SQL statement, please check your syntax closely. If you can't find something wrong with it, it's extremely likely that your current version of MySQL Server doesn't support the syntax you are using. If you are using the current version and the manual doesn't cover the syntax that you are using, MySQL Server doesn't support your statement. In this case, your options are to implement the syntax yourself or email [<licensing@mysql.com>](mailto:licensing@mysql.com) and ask for an offer to implement it.

If the manual covers the syntax you are using, but you have an older version of MySQL Server, you should check the MySQL change history to see when the syntax was implemented. In this case, you have the option of upgrading to a newer version of MySQL Server.

- For solutions to some common problems, see [Section B.1, “Problems and Common Errors”](#).
- Search the bugs database at <http://bugs.mysql.com/> to see whether the bug has been reported and fixed.
- Search the MySQL mailing list archives at <http://lists.mysql.com/>. See [Section 1.7.1, “MySQL Mailing Lists”](#).
- You can also use <http://www.mysql.com/search/> to search all the Web pages (including the manual) that are located at the MySQL AB Web site.

If you can't find an answer in the manual, the bugs database, or the mailing list archives, check with your local MySQL expert. If you still can't find an answer to your question, please use the following guidelines for reporting the bug.

The normal way to report bugs is to visit <http://bugs.mysql.com/>, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports. If you have no Web access, you can generate a bug report by using the `mysqlbug` script described at the end of this section.

Bugs posted in the bugs database at <http://bugs.mysql.com/> that are corrected for a given release are noted in the change history.

If you have found a sensitive security bug in MySQL, you can send email to [<security@mysql.com>](mailto:security@mysql.com).

To discuss problems with other users, you can use one of the MySQL mailing lists. [Section 1.7.1, “MySQL Mailing Lists”](#).

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release. This section helps you write your report correctly so that you don't waste your time doing things that may not help us much or at all. Please read this section carefully and make sure that all the information described here is included in your report.

Preferably, you should test the problem using the latest production or development version of MySQL Server before posting. Anyone should be able to repeat the bug by just using `mysql test < script_file` on your test case or by running the shell or Perl script that you include in the bug report. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

It is most helpful when a good description of the problem is included in the bug report. That is, give a good example of everything you did that led to the problem and describe, in exact detail, the problem itself. The best reports are those that include a full example showing how to reproduce the bug or problem. See [MySQL Internals: Porting](#).

Remember that it is possible for us to respond to a report containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details don't matter. A good principle to follow is that if you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of the MySQL distribution that you use, and (b) not fully describing the platform on which the MySQL server is installed (including the platform type and version number). These are highly relevant pieces of information, and in 99 cases out of 100, the bug report is useless without them. Very often we get questions like, "Why doesn't this work for me?" Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has been fixed in newer MySQL versions. Errors often are platform-dependent. In such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If you compiled MySQL from source, remember also to provide information about your compiler if it is related to the problem. Often people find bugs in compilers and think the problem is MySQL-related. Most compilers are under development all the time and become better version by version. To determine whether your problem depends on your compiler, we need to know what compiler you used. Note that every compiling problem should be regarded as a bug and reported accordingly.

If a program produces an error message, it is very important to include the message in your report. If we try to search for something from the archives, it is better that the error message reported exactly matches the one that the program produces. (Even the lettercase should be observed.) It is best to copy and paste the entire error message into your report. You should never try to reproduce the message from memory.

If you have a problem with Connector/ODBC (MyODBC), please try to generate a trace file and send it with your report. See the MyODBC section of [Chapter 26, Connectors](#).

If your report includes long query output lines from test cases that you run with the `mysql` command-line tool, you can make the output more readable by using the `--vertical` option or the `\G` statement terminator. The [EXPLAIN SELECT](#) example later in this section demonstrates the use of `\G`.

Please include the following information in your report:

- The version number of the MySQL distribution you are using (for example, MySQL 5.0.19). You can find out which version you are running by executing `mysqladmin version`. The `mysqladmin` program can be found in the `bin` directory under your MySQL installation directory.
- The manufacturer and model of the machine on which you experience the problem.
- The operating system name and version. If you work with Windows, you can usually get the name and version number by double-clicking your My Computer icon and pulling down the "Help/About Windows" menu. For most Unix-like operating systems, you can get this information by executing the command `uname -a`.
- Sometimes the amount of memory (real and virtual) is relevant. If in doubt, include these values.
- If you are using a source distribution of the MySQL software, include the name and version number of the compiler that you used. If you have a binary distribution, include the distribution name.
- If the problem occurs during compilation, include the exact error messages and also a few lines of context around the offending code in the file where the error occurs.
- If `mysqld` died, you should also report the statement that crashed `mysqld`. You can usually get this information by running `mysqld` with query logging enabled, and then looking in the log after `mysqld` crashes. See [MySQL Internals: Porting](#).
- If a database table is related to the problem, include the output from the `SHOW CREATE TABLE db_name.tbl_name` statement in the bug report. This is a very easy way to get the definition of any table in a database. The information helps us create a situation matching the one that you have experienced.
- The SQL mode in effect when the problem occurred can be significant, so please report the value of the `sql_mode` system vari-

able. For stored procedure, stored function, and trigger objects, the relevant `sql_mode` value is the one in effect when the object was created. For a stored procedure or function, the `SHOW CREATE PROCEDURE` or `SHOW CREATE FUNCTION` statement shows the relevant SQL mode, or you can query `INFORMATION_SCHEMA` for the information:

```
SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE
FROM INFORMATION_SCHEMA.ROUTINES;
```

For triggers, you can use this statement:

```
SELECT EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, TRIGGER_NAME, SQL_MODE
FROM INFORMATION_SCHEMA.TRIGGERS;
```

- For performance-related bugs or problems with `SELECT` statements, you should always include the output of `EXPLAIN SELECT ...`, and at least the number of rows that the `SELECT` statement produces. You should also include the output from `SHOW CREATE TABLE tbl_name` for each table that is involved. The more information you provide about your situation, the more likely it is that someone can help you.

The following is an example of a very good bug report. The statements are run using the `mysql` command-line tool. Note the use of the `\G` statement terminator for statements that would otherwise provide very long output lines that are difficult to read.

```
mysql> SHOW VARIABLES;
mysql> SHOW COLUMNS FROM ... \G
<output from SHOW COLUMNS>
mysql> EXPLAIN SELECT ... \G
<output from EXPLAIN>
mysql> FLUSH STATUS;
mysql> SELECT ...;
<A short version of the output from SELECT,
including the time taken to run the query>
mysql> SHOW STATUS;
<output from SHOW STATUS>
```

- If a bug or problem occurs while running `mysqld`, try to provide an input script that reproduces the anomaly. This script should include any necessary source files. The more closely the script can reproduce your situation, the better. If you can make a reproducible test case, you should upload it to be attached to the bug report.

If you can't provide a script, you should at least include the output from `mysqladmin variables extended-status processlist` in your report to provide some information on how your system is performing.

- If you can't produce a test case with only a few rows, or if the test table is too big to be included in the bug report (more than 10 rows), you should dump your tables using `mysqldump` and create a `README` file that describes your problem. Create a compressed archive of your files using `tar` and `gzip` or `zip`, and use FTP to transfer the archive to <ftp://ftp.mysql.com/pub/mysql/upload/>. Then enter the problem into our bugs database at <http://bugs.mysql.com/>.
- If you believe that the MySQL server produces a strange result from a statement, include not only the result, but also your opinion of what the result should be, and an explanation describing the basis for your opinion.
- When you provide an example of the problem, it's better to use the table names, variable names, and so forth that exist in your actual situation than to come up with new names. The problem could be related to the name of a table or variable. These cases are rare, perhaps, but it is better to be safe than sorry. After all, it should be easier for you to provide an example that uses your actual situation, and it is by all means better for us. If you have data that you don't want to be visible to others in the bug report, you can use FTP to transfer it to <ftp://ftp.mysql.com/pub/mysql/upload/>. If the information is really top secret and you don't want to show it even to us, go ahead and provide an example using other names, but please regard this as the last choice.
- Include all the options given to the relevant programs, if possible. For example, indicate the options that you use when you start the `mysqld` server, as well as the options that you use to run any MySQL client programs. The options to programs such as `mysqld` and `mysql`, and to the `configure` script, are often key to resolving problems and are very relevant. It is never a bad idea to include them. If your problem involves a program written in a language such as Perl or PHP, please include the language processor's version number, as well as the version for any modules that the program uses. For example, if you have a Perl script that uses the `DBI` and `DBD: :mysql` modules, include the version numbers for Perl, `DBI`, and `DBD: :mysql`.
- If your question is related to the privilege system, please include the output of `mysqlaccess`, the output of `mysqladmin reload`, and all the error messages you get when trying to connect. When you test your privileges, you should first run `mysqlaccess`. After this, execute `mysqladmin reload version` and try to connect with the program that gives you trouble. `mysqlaccess` can be found in the `bin` directory under your MySQL installation directory.
- If you have a patch for a bug, do include it. But don't assume that the patch is all we need, or that we can use it, if you don't provide

some necessary information such as test cases showing the bug that your patch fixes. We might find problems with your patch or we might not understand it at all. If so, we can't use it.

If we can't verify the exact purpose of the patch, we won't use it. Test cases help us here. Show that the patch handles all the situations that may occur. If we find a borderline case (even a rare one) where the patch won't work, it may be useless.

- Guesses about what the bug is, why it occurs, or what it depends on are usually wrong. Even the MySQL team can't guess such things without first using a debugger to determine the real cause of a bug.
- Indicate in your bug report that you have checked the reference manual and mail archive so that others know you have tried to solve the problem yourself.
- If the problem is that your data appears corrupt or you get errors when you access a particular table, you should first check your tables and then try to repair them with `CHECK TABLE` and `REPAIR TABLE` or with `myisamchk`. See [Chapter 5, Database Administration](#).

If you are running Windows, please verify the value of `lower_case_table_names` using the `SHOW VARIABLES LIKE 'lower_case_table_names'` command. This variable affects how the server handles lettercase of database and table names. Its effect for a given value should be as described in [Section 8.2.2, “Identifier Case Sensitivity”](#).

- If you often get corrupted tables, you should try to find out when and why this happens. In this case, the error log in the MySQL data directory may contain some information about what happened. (This is the file with the `.err` suffix in the name.) See [Section 5.11.2, “The Error Log”](#). Please include any relevant information from this file in your bug report. Normally `mysqld` should *never* crash a table if nothing killed it in the middle of an update. If you can find the cause of `mysqld` dying, it's much easier for us to provide you with a fix for the problem. See [Section B.1.1, “How to Determine What Is Causing a Problem”](#).
- If possible, download and install the most recent version of MySQL Server and check whether it solves your problem. All versions of the MySQL software are thoroughly tested and should work without problems. We believe in making everything as backward-compatible as possible, and you should be able to switch MySQL versions without difficulty. See [Section 2.1.2, “Choosing Which MySQL Distribution to Install”](#).

If you have no Web access and cannot report a bug by visiting <http://bugs.mysql.com/>, you can use the `mysqlbug` script to generate a bug report (or a report about any problem). `mysqlbug` helps you generate a report by determining much of the following information automatically, but if something important is missing, please include it with your message. `mysqlbug` can be found in the `scripts` directory (source distribution) and in the `bin` directory under your MySQL installation directory (binary distribution).

1.9. MySQL Standards Compliance

This section describes how MySQL relates to the ANSI/ISO SQL standards. MySQL Server has many extensions to the SQL standard, and here you can find out what they are and how to use them. You can also find information about functionality missing from MySQL Server, and how to work around some of the differences.

The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, and “SQL:2003” refers to the current version of the standard. We use the phrase “the SQL standard” or “standard SQL” to mean the current version of the SQL Standard at any time.

One of our main goals with the product is to continue to work toward compliance with the SQL standard, but without sacrificing speed or reliability. We are not afraid to add extensions to SQL or support for non-SQL features if this greatly increases the usability of MySQL Server for a large segment of our user base. The `HANDLER` interface is an example of this strategy. See [Section 12.2.3, “HANDLER Syntax”](#).

We continue to support transactional and non-transactional databases to satisfy both mission-critical 24/7 usage and heavy Web or logging usage.

MySQL Server was originally designed to work with medium-sized databases (10-100 million rows, or about 100MB per table) on small computer systems. Today MySQL Server handles terabyte-sized databases, but the code can also be compiled in a reduced version suitable for hand-held and embedded devices. The compact design of the MySQL server makes development in both directions possible without any conflicts in the source tree.

Currently, we are not targeting real-time support, although MySQL replication capabilities offer significant functionality.

MySQL supports high-availability database clustering using the `NDBCluster` storage engine. See [Chapter 16, MySQL Cluster](#).

We are implementing XML functionality beginning in MySQL 5.1, which supports most of the W3C XPath standard. We plan to in-

crease support for XML as part of future MySQL development. See [Section 11.10, “XML Functions”](#).

1.9.1. What Standards MySQL Follows

Our aim is to support the full ANSI/ISO SQL standard, but without making concessions to speed and quality of the code.

ODBC levels 0-3.51.

1.9.2. Selecting SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differentially for different clients. This capability enables each application to tailor the server's operating mode to its own requirements.

SQL modes control aspects of server operation such as what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

You can set the default SQL mode by starting `mysqld` with the `--sql-mode="mode_value"` option. You can also change the mode at runtime by setting the `sql_mode` system variable with a `SET [SESSION|GLOBAL] sql_mode='mode_value'` statement.

For more information on setting the SQL mode, see [Section 5.2.6, “SQL Modes”](#).

1.9.3. Running MySQL in ANSI Mode

You can tell `mysqld` to run in ANSI mode with the `--ansi` startup option. Running the server in ANSI mode is the same as starting it with the following options:

```
--transaction-isolation=SERIALIZABLE --sql-mode=ANSI
```

You can achieve the same effect at runtime by executing these two statements:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
SET GLOBAL sql_mode = 'ANSI';
```

You can see that setting the `sql_mode` system variable to `'ANSI'` enables all SQL mode options that are relevant for ANSI mode as follows:

```
mysql> SET GLOBAL sql_mode='ANSI';  
mysql> SELECT @@global.sql_mode;  
-> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI'
```

Note that running the server in ANSI mode with `--ansi` is not quite the same as setting the SQL mode to `'ANSI'`. The `--ansi` option affects the SQL mode and also sets the transaction isolation level. Setting the SQL mode to `'ANSI'` has no effect on the isolation level.

See [Section 5.2.2, “Command Options”](#), and [Section 1.9.2, “Selecting SQL Modes”](#).

1.9.4. MySQL Extensions to Standard SQL

MySQL Server supports some extensions that you probably won't find in other SQL DBMSs. Be warned that if you use them, your code won't be portable to other SQL servers. In some cases, you can write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```


If you add a version number after the ‘!’ character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The following descriptions list MySQL extensions, organized by category.

- Organization of data on disk

MySQL Server maps each database to a directory under the MySQL data directory, and maps tables within a database to filenames in the database directory. This has a few implications:

- Database and table names are case sensitive in MySQL Server on operating systems that have case-sensitive filenames (such as most Unix systems). See [Section 8.2.2, “Identifier Case Sensitivity”](#).
- You can use standard system commands to back up, rename, move, delete, and copy tables that are managed by the `MyISAM` storage engine. For example, it is possible to rename a `MyISAM` table by renaming the `.MYD`, `.MYI`, and `.frm` files to which the table corresponds. (Nevertheless, it is preferable to use `RENAME TABLE` or `ALTER TABLE ... RENAME` and let the server rename the files.)

Database and table names cannot contain pathname separator characters (`/`, `\`).

- General language syntax

- By default, strings can be enclosed by either `""` or `' '`, not just by `' '`. (If the `ANSI_QUOTES` SQL mode is enabled, strings can be enclosed only by `' '` and the server interprets strings enclosed by `""` as identifiers.)
- `'\'` is the escape character in strings.
- In SQL statements, you can access tables from different databases with the `db_name.tbl_name` syntax. Some SQL servers provide the same functionality but call this `User space`. MySQL Server doesn't support tablespaces such as used in statements like this: `CREATE TABLE ralph.my_table ... IN my_tablespace`.

- SQL statement syntax

- The `ANALYZE TABLE`, `CHECK TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.
- The `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE` statements. See [Section 12.1.6, “CREATE DATABASE Syntax”](#), [Section 12.1.12, “DROP DATABASE Syntax”](#), and [Section 12.1.1, “ALTER DATABASE Syntax”](#).
- The `DO` statement.
- `EXPLAIN SELECT` to obtain a description of how tables are processed by the query optimizer.
- The `FLUSH` and `RESET` statements.
- The `SET` statement. See [Section 12.5.3, “SET Syntax”](#).
- The `SHOW` statement. See [Section 12.5.4, “SHOW Syntax”](#). As of MySQL 5.0, the information produced by many of the MySQL-specific `SHOW` statements can be obtained in more standard fashion by using `SELECT` to query `INFORMATION_SCHEMA`. See [Chapter 23, The INFORMATION_SCHEMA Database](#).
- Use of `LOAD DATA INFILE`. In many cases, this syntax is compatible with Oracle's `LOAD DATA INFILE`. See [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).
- Use of `RENAME TABLE`. See [Section 12.1.19, “RENAME TABLE Syntax”](#).
- Use of `REPLACE` instead of `DELETE` plus `INSERT`. See [Section 12.2.6, “REPLACE Syntax”](#).
- Use of `CHANGE col_name`, `DROP col_name`, or `DROP INDEX, IGNORE` or `RENAME` in `ALTER TABLE` statements. Use of multiple `ADD`, `ALTER`, `DROP`, or `CHANGE` clauses in an `ALTER TABLE` statement. See [Section 12.1.4, “ALTER TABLE Syntax”](#).
- Use of index names, indexes on a prefix of a column, and use of `INDEX` or `KEY` in `CREATE TABLE` statements. See [Sec-](#)

tion 12.1.10, “CREATE TABLE Syntax”.

- Use of `TEMPORARY` or `IF NOT EXISTS` with `CREATE TABLE`.
- Use of `IF EXISTS` with `DROP TABLE` and `DROP DATABASE`.
- The capability of dropping multiple tables with a single `DROP TABLE` statement.
- The `ORDER BY` and `LIMIT` clauses of the `UPDATE` and `DELETE` statements.
- `INSERT INTO tbl_name SET col_name = ...` syntax.
- The `DELAYED` clause of the `INSERT` and `REPLACE` statements.
- The `LOW_PRIORITY` clause of the `INSERT`, `REPLACE`, `DELETE`, and `UPDATE` statements.
- Use of `INTO OUTFILE` or `INTO DUMPFILE` in `SELECT` statements. See Section 12.2.7, “SELECT Syntax”.
- Options such as `STRAIGHT_JOIN` or `SQL_SMALL_RESULT` in `SELECT` statements.
- You don't need to name all selected columns in the `GROUP BY` clause. This gives better performance for some very specific, but quite normal queries. See Section 11.12, “Functions and Modifiers for Use with GROUP BY Clauses”.
- You can specify `ASC` and `DESC` with `GROUP BY`, not just with `ORDER BY`.
- The ability to set variables in a statement with the `:=` assignment operator:

```
mysql> SELECT @a:=SUM(total),@b:=COUNT(*),@a/@b AS avg
-> FROM test_table;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
```

- Data types
 - The `MEDIUMINT`, `SET`, and `ENUM` data types, and the various `BLOB` and `TEXT` data types.
 - The `AUTO_INCREMENT`, `BINARY`, `NULL`, `UNSIGNED`, and `ZEROFILL` data type attributes.
- Functions and operators
 - To make it easier for users who migrate from other SQL environments, MySQL Server supports aliases for many functions. For example, all string functions support both standard SQL syntax and ODBC syntax.
 - MySQL Server understands the `||` and `&&` operators to mean logical OR and AND, as in the C programming language. In MySQL Server, `||` and `OR` are synonyms, as are `&&` and `AND`. Because of this nice syntax, MySQL Server doesn't support the standard SQL `||` operator for string concatenation; use `CONCAT()` instead. Because `CONCAT()` takes any number of arguments, it's easy to convert use of the `||` operator to MySQL Server.
 - Use of `COUNT(DISTINCT value_list)` where `value_list` has more than one element.
 - String comparisons are case-insensitive by default, with sort ordering determined by the collation of the current character set, which is `latin1` (cp1252 West European) by default. If you don't like this, you should declare your columns with the `BINARY` attribute or use the `BINARY` cast, which causes comparisons to be done using the underlying character code values rather than a lexical ordering.
 - The `%` operator is a synonym for `MOD()`. That is, `N % M` is equivalent to `MOD(N,M)`. `%` is supported for C programmers and for compatibility with PostgreSQL.
 - The `=`, `<>`, `<=<`, `>=>`, `<<`, `>>`, `<=>`, `AND`, `OR`, or `LIKE` operators may be used in expressions in the output column list (to the left of the `FROM`) in `SELECT` statements. For example:

```
mysql> SELECT col1=1 AND col2=2 FROM my_table;
```

- The `LAST_INSERT_ID()` function returns the most recent `AUTO_INCREMENT` value. See Section 11.11.3, “Information Functions”.

- [LIKE](#) is allowed on numeric values.
- The [REGEXP](#) and [NOT REGEXP](#) extended regular expression operators.
- [CONCAT\(\)](#) or [CHAR\(\)](#) with one argument or more than two arguments. (In MySQL Server, these functions can take a variable number of arguments.)
- The [BIT_COUNT\(\)](#), [CASE](#), [ELT\(\)](#), [FROM_DAYS\(\)](#), [FORMAT\(\)](#), [IF\(\)](#), [PASSWORD\(\)](#), [ENCRYPT\(\)](#), [MD5\(\)](#), [ENCODE\(\)](#), [DECODE\(\)](#), [PERIOD_ADD\(\)](#), [PERIOD_DIFF\(\)](#), [TO_DAYS\(\)](#), and [WEEKDAY\(\)](#) functions.
- Use of [TRIM\(\)](#) to trim substrings. Standard SQL supports removal of single characters only.
- The [GROUP BY](#) functions [STD\(\)](#), [BIT_OR\(\)](#), [BIT_AND\(\)](#), [BIT_XOR\(\)](#), and [GROUP_CONCAT\(\)](#). See [Section 11.12](#), “Functions and Modifiers for Use with GROUP BY Clauses”.

For a prioritized list indicating when new extensions are added to MySQL Server, you should consult the online MySQL development roadmap at <http://dev.mysql.com/doc/mysql/en/roadmap.html>.

1.9.5. MySQL Differences from Standard SQL

We try to make MySQL Server follow the ANSI SQL standard and the ODBC SQL standard, but MySQL Server performs operations differently in some cases:

- There are several differences between the MySQL and standard SQL privilege systems. For example, in MySQL, privileges for a table are not automatically revoked when you delete a table. You must explicitly issue a [REVOKE](#) statement to revoke privileges for a table. For more information, see [Section 12.5.1.5](#), “[REVOKE Syntax](#)”.
- The [CAST\(\)](#) function does not support cast to [REAL](#) or [BIGINT](#). See [Section 11.9](#), “[Cast Functions and Operators](#)”.

1.9.5.1. SELECT INTO TABLE

MySQL Server doesn't support the [SELECT ... INTO TABLE](#) Sybase SQL extension. Instead, MySQL Server supports the [INSERT INTO ... SELECT](#) standard SQL syntax, which is basically the same thing. See [Section 12.2.4.1](#), “[INSERT ... SELECT Syntax](#)”. For example:

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

Alternatively, you can use [SELECT ... INTO OUTFILE](#) or [CREATE TABLE ... SELECT](#).

As of MySQL 5.0, you can use [SELECT ... INTO](#) with user-defined variables. The same syntax can also be used inside stored routines using cursors and local variables. See [Section 19.2.7.3](#), “[SELECT ... INTO Statement](#)”.

1.9.5.2. Transactions and Atomic Operations

MySQL Server (version 3.23-max and all versions 4.0 and above) supports transactions with the [InnoDB](#) transactional storage engine. [InnoDB](#) provides *full ACID* compliance. See [Chapter 13](#), [Storage Engines](#). For information about [InnoDB](#) differences from standard SQL with regard to treatment of transaction errors, see [Section 13.5.15](#), “[InnoDB Error Handling](#)”.

The other non-transactional storage engines in MySQL Server (such as [MyISAM](#)) follow a different paradigm for data integrity called “atomic operations.” In transactional terms, [MyISAM](#) tables effectively always operate in [AUTOCOMMIT=1](#) mode. Atomic operations often offer comparable integrity with higher performance.

Because MySQL Server supports both paradigms, you can decide whether your applications are best served by the speed of atomic operations or the use of transactional features. This choice can be made on a per-table basis.

As noted, the trade-off for transactional versus non-transactional storage engines lies mostly in performance. Transactional tables have significantly higher memory and disk space requirements, and more CPU overhead. On the other hand, transactional storage engines such as [InnoDB](#) also offer many significant features. MySQL Server's modular design allows the concurrent use of different storage engines to suit different requirements and deliver optimum performance in all situations.

MySQL Enterprise

For expert advice on choosing and tuning storage engines, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

But how do you use the features of MySQL Server to maintain rigorous integrity even with the non-transactional **MyISAM** tables, and how do these features compare with the transactional storage engines?

- If your applications are written in a way that is dependent on being able to call **ROLLBACK** rather than **COMMIT** in critical situations, transactions are more convenient. Transactions also ensure that unfinished updates or corrupting activities are not committed to the database; the server is given the opportunity to do an automatic rollback and your database is saved.

If you use non-transactional tables, MySQL Server in almost all cases allows you to resolve potential problems by including simple checks before updates and by running simple scripts that check the databases for inconsistencies and automatically repair or warn if such an inconsistency occurs. Note that just by using the MySQL log or even adding one extra log, you can normally fix tables perfectly with no data integrity loss.

- More often than not, critical transactional updates can be rewritten to be atomic. Generally speaking, all integrity problems that transactions solve can be done with **LOCK TABLES** or atomic updates, ensuring that there are no automatic aborts from the server, which is a common problem with transactional database systems.
- To be safe with MySQL Server, regardless of whether you use transactional tables, you only need to have backups and have binary logging turned on. When that is true, you can recover from any situation that you could with any other transactional database system. It is always good to have backups, regardless of which database system you use.

The transactional paradigm has its benefits and its drawbacks. Many users and application developers depend on the ease with which they can code around problems where an abort appears to be necessary, or is necessary. However, even if you are new to the atomic operations paradigm, or more familiar with transactions, do consider the speed benefit that non-transactional tables can offer on the order of three to five times the speed of the fastest and most optimally tuned transactional tables.

In situations where integrity is of highest importance, MySQL Server offers transaction-level reliability and integrity even for non-transactional tables. If you lock tables with **LOCK TABLES**, all updates stall until integrity checks are made. If you obtain a **READ LOCAL** lock (as opposed to a write lock) for a table that allows concurrent inserts at the end of the table, reads are allowed, as are inserts by other clients. The newly inserted records are not be seen by the client that has the read lock until it releases the lock. With **INSERT DELAYED**, you can write inserts that go into a local queue until the locks are released, without having the client wait for the insert to complete. See [Section 6.3.3, “Concurrent Inserts”](#), and [Section 12.2.4.2, “INSERT DELAYED Syntax”](#).

“Atomic,” in the sense that we mean it, is nothing magical. It only means that you can be sure that while each specific update is running, no other user can interfere with it, and there can never be an automatic rollback (which can happen with transactional tables if you are not very careful). MySQL Server also guarantees that there are no dirty reads.

Following are some techniques for working with non-transactional tables:

- Loops that need transactions normally can be coded with the help of **LOCK TABLES**, and you don't need cursors to update records on the fly.
- To avoid using **ROLLBACK**, you can employ the following strategy:
 1. Use **LOCK TABLES** to lock all the tables you want to access.
 2. Test the conditions that must be true before performing the update.
 3. Update if the conditions are satisfied.
 4. Use **UNLOCK TABLES** to release your locks.

This is usually a much faster method than using transactions with possible rollbacks, although not always. The only situation this solution doesn't handle is when someone kills the threads in the middle of an update. In that case, all locks are released but some of the updates may not have been executed.

- You can also use functions to update records in a single operation. You can get a very efficient application by using the following techniques:
 - Modify columns relative to their current value.

- Update only those columns that actually have changed.

For example, when we are updating customer information, we update only the customer data that has changed and test only that none of the changed data, or data that depends on the changed data, has changed compared to the original row. The test for changed data is done with the [WHERE](#) clause in the [UPDATE](#) statement. If the record wasn't updated, we give the client a message: “Some of the data you have changed has been changed by another user.” Then we show the old row versus the new row in a window so that the user can decide which version of the customer record to use.

This gives us something that is similar to column locking but is actually even better because we only update some of the columns, using values that are relative to their current values. This means that typical [UPDATE](#) statements look something like these:

```
UPDATE tablename SET pay_back=pay_back+125;

UPDATE customer
SET
  customer_date='current_date',
  address='new address',
  phone='new phone',
  money_owed_to_us=money_owed_to_us-125
WHERE
  customer_id=id AND address='old address' AND phone='old phone';
```

This is very efficient and works even if another client has changed the values in the [pay_back](#) or [money_owed_to_us](#) columns.

- In many cases, users have wanted [LOCK TABLES](#) or [ROLLBACK](#) for the purpose of managing unique identifiers. This can be handled much more efficiently without locking or rolling back by using an [AUTO_INCREMENT](#) column and either the [LAST_INSERT_ID\(\)](#) SQL function or the [mysql_insert_id\(\)](#) C API function. See [Section 11.11.3, “Information Functions”](#), and [Section 25.2.3.37, “mysql_insert_id\(\)”](#).

You can generally code around the need for row-level locking. Some situations really do need it, and [InnoDB](#) tables support row-level locking. Otherwise, with [MyISAM](#) tables, you can use a flag column in the table and do something like the following:

```
UPDATE tbl_name SET row_flag=1 WHERE id=ID;
```

MySQL returns [1](#) for the number of affected rows if the row was found and [row_flag](#) wasn't [1](#) in the original row. You can think of this as though MySQL Server changed the preceding statement to:

```
UPDATE tbl_name SET row_flag=1 WHERE id=ID AND row_flag <> 1;
```

1.9.5.3. Stored Routines and Triggers

Stored procedures and functions are implemented beginning with MySQL 5.0. See [Chapter 19, *Stored Procedures and Functions*](#).

Basic trigger functionality is implemented beginning with MySQL 5.0.2, with further development planned for MySQL 5.1. See [Chapter 20, *Triggers*](#).

1.9.5.4. Foreign Keys

In MySQL Server 3.23.44 and up, the [InnoDB](#) storage engine supports checking of foreign key constraints, including [CASCADE](#), [ON DELETE](#), and [ON UPDATE](#). See [Section 13.5.6.4, “FOREIGN KEY Constraints”](#).

For storage engines other than [InnoDB](#), MySQL Server parses the [FOREIGN KEY](#) syntax in [CREATE TABLE](#) statements, but does not use or store it. In the future, the implementation will be extended to store this information in the table specification file so that it may be retrieved by [mysqldump](#) and ODBC. At a later stage, foreign key constraints will be implemented for [MyISAM](#) tables as well.

Foreign key enforcement offers several benefits to database developers:

- Assuming proper design of the relationships, foreign key constraints make it more difficult for a programmer to introduce an inconsistency into the database.
- Centralized checking of constraints by the database server makes it unnecessary to perform these checks on the application side. This eliminates the possibility that different applications may not all check the constraints in the same way.

- Using cascading updates and deletes can simplify the application code.
- Properly designed foreign key rules aid in documenting relationships between tables.

Do keep in mind that these benefits come at the cost of additional overhead for the database server to perform the necessary checks. Additional checking by the server affects performance, which for some applications may be sufficiently undesirable as to be avoided if possible. (Some major commercial applications have coded the foreign key logic at the application level for this reason.)

MySQL gives database developers the choice of which approach to use. If you don't need foreign keys and want to avoid the overhead associated with enforcing referential integrity, you can choose another storage engine instead, such as [MyISAM](#). (For example, the [MyISAM](#) storage engine offers very fast performance for applications that perform only [INSERT](#) and [SELECT](#) operations. In this case, the table has no holes in the middle and the inserts can be performed concurrently with retrievals. See [Section 6.3.3, “Concurrent Inserts”](#).)

If you choose not to take advantage of referential integrity checks, keep the following considerations in mind:

- In the absence of server-side foreign key relationship checking, the application itself must handle relationship issues. For example, it must take care to insert rows into tables in the proper order, and to avoid creating orphaned child records. It must also be able to recover from errors that occur in the middle of multiple-record insert operations.
- If [ON DELETE](#) is the only referential integrity capability an application needs, you can achieve a similar effect as of MySQL Server 4.0 by using multiple-table [DELETE](#) statements to delete rows from many tables with a single statement. See [Section 12.2.1, “DELETE Syntax”](#).
- A workaround for the lack of [ON DELETE](#) is to add the appropriate [DELETE](#) statements to your application when you delete records from a table that has a foreign key. In practice, this is often as quick as using foreign keys and is more portable.

Be aware that the use of foreign keys can sometimes lead to problems:

- Foreign key support addresses many referential integrity issues, but it is still necessary to design key relationships carefully to avoid circular rules or incorrect combinations of cascading deletes.
- It is not uncommon for a DBA to create a topology of relationships that makes it difficult to restore individual tables from a backup. (MySQL alleviates this difficulty by allowing you to temporarily disable foreign key checks when reloading a table that depends on other tables. See [Section 13.5.6.4, “FOREIGN KEY Constraints”](#). As of MySQL 4.1.1, [mysqldump](#) generates dump files that take advantage of this capability automatically when they are reloaded.)

Note that foreign keys in SQL are used to check and enforce referential integrity, not to join tables. If you want to get results from multiple tables from a [SELECT](#) statement, you do this by performing a join between them:

```
SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

See [Section 12.2.7.1, “JOIN Syntax”](#), and [Section 3.6.6, “Using Foreign Keys”](#).

The [FOREIGN KEY](#) syntax without [ON DELETE . . .](#) is often used by ODBC applications to produce automatic [WHERE](#) clauses.

1.9.5.5. Views

Views (including updatable views) are implemented beginning with MySQL Server 5.0.1. See [Chapter 22, Views](#).

Views are useful for allowing users to access a set of relations (tables) as if it were a single table, and limiting their access to just that. Views can also be used to restrict access to rows (a subset of a particular table). For access control to columns, you can also use the sophisticated privilege system in MySQL Server. See [Section 5.7, “The MySQL Access Privilege System”](#).

In designing an implementation of views, our ambitious goal, as much as is possible within the confines of SQL, has been full compliance with “Codd's Rule #6” for relational database systems: “All views that are theoretically updatable, should in practice also be updatable.”

1.9.5.6. '-' as the Start of a Comment

Standard SQL uses the C syntax `/* this is a comment */` for comments, and MySQL Server supports this syntax as well.

MySQL also support extensions to this syntax that allow MySQL-specific SQL to be embedded in the comment, as described in [Section 8.5, “Comment Syntax”](#).

Standard SQL uses ‘--’ as a start-comment sequence. MySQL Server uses ‘#’ as the start comment character. MySQL Server 3.23.3 and up also supports a variant of the ‘--’ comment style. That is, the ‘--’ start-comment sequence must be followed by a space (or by a control character such as a newline). The space is required to prevent problems with automatically generated SQL queries that use constructs such as the following, where we automatically insert the value of the payment for `payment`:

```
UPDATE account SET credit=credit-payment
```

Consider about what happens if `payment` has a negative value such as `-1`:

```
UPDATE account SET credit=credit--1
```

`credit--1` is a legal expression in SQL, but ‘--’ is interpreted as the start of a comment, part of the expression is discarded. The result is a statement that has a completely different meaning than intended:

```
UPDATE account SET credit=credit
```

The statement produces no change in value at all. This illustrates that allowing comments to start with ‘--’ can have serious consequences.

Using our implementation requires a space following the ‘--’ in order for it to be recognized as a start-comment sequence in MySQL Server 3.23.3 and newer. Therefore, `credit--1` is safe to use.

Another safe feature is that the `mysql` command-line client ignores lines that start with ‘--’.

The following information is relevant only if you are running a MySQL version earlier than 3.23.3:

If you have an SQL script in a text file that contains ‘--’ comments, you should use the `replace` utility as follows to convert the comments to use ‘#’ characters before executing the script:

```
shell> replace " --" " #" < text-file-with-funny-comments.sql \  
| mysql db_name
```

That is safer than executing the script in the usual way:

```
shell> mysql db_name < text-file-with-funny-comments.sql
```

You can also edit the script file “in place” to change the ‘--’ comments to ‘#’ comments:

```
shell> replace " --" " #" -- text-file-with-funny-comments.sql
```

Change them back with this command:

```
shell> replace " #" " --" -- text-file-with-funny-comments.sql
```

See [Section 7.26, “replace — A String-Replacement Utility”](#).

1.9.6. How MySQL Deals with Constraints

MySQL allows you to work both with transactional tables that allow rollback and with non-transactional tables that do not. Because of this, constraint handling is a bit different in MySQL than in other DBMSs. We must handle the case when you have inserted or updated a lot of rows in a non-transactional table for which changes cannot be rolled back when an error occurs.

The basic philosophy is that MySQL Server tries to produce an error for anything that it can detect while parsing a statement to be executed, and tries to recover from any errors that occur while executing the statement. We do this in most cases, but not yet for all.

The options MySQL has when an error occurs are to stop the statement in the middle or to recover as well as possible from the problem and continue. By default, the server follows the latter course. This means, for example, that the server may coerce illegal values to the closest legal values.

Several SQL mode options are available to provide greater control over handling of bad data values and whether to continue statement

execution or abort when errors occur. Using these options, you can configure MySQL Server to act in a more traditional fashion that is like other DBMSs that reject improper input. The SQL mode can be set globally at server startup to affect all clients. Individual clients can set the SQL mode at runtime, which enables each client to select the behavior most appropriate for its requirements. See [Section 5.2.6, “SQL Modes”](#).

MySQL Enterprise

To be alerted when there is no form of server-enforced data integrity, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

The following sections describe how MySQL Server handles different types of constraints.

1.9.6.1. PRIMARY KEY and UNIQUE Index Constraints

Normally, an error occurs when you try to **INSERT** or **UPDATE** a row that causes a primary key, unique key, or foreign key violation. If you are using a transactional storage engine such as **InnoDB**, MySQL automatically rolls back the statement. If you are using a non-transactional storage engine, MySQL stops processing the statement at the row for which the error occurred and leaves any remaining rows unprocessed.

If you want to ignore such key violations, MySQL supports an **IGNORE** keyword for **INSERT** and **UPDATE**. In this case, MySQL ignores any key violations and continues processing with the next row. See [Section 12.2.4, “INSERT Syntax”](#), and [Section 12.2.10, “UPDATE Syntax”](#).

You can get information about the number of rows actually inserted or updated with the `mysql_info()` C API function. You can also use the **SHOW WARNINGS** statement. See [Section 25.2.3.35, “mysql_info\(\)”](#), and [Section 12.5.4.32, “SHOW WARNINGS Syntax”](#).

Currently, only **InnoDB** tables support foreign keys. See [Section 13.5.6.4, “FOREIGN KEY Constraints”](#). Foreign key support in **MyISAM** tables is scheduled for implementation in MySQL 5.2. See [Section 1.6, “MySQL Development Roadmap”](#).

1.9.6.2. Constraints on Invalid Data

By default, MySQL is forgiving of illegal or improper data values and coerces them to legal values for data entry. However, you can change the server SQL mode to select more traditional treatment of bad values such that the server rejects them and aborts the statement in which they occur. See [Section 5.2.6, “SQL Modes”](#).

This section describes the default (forgiving) behavior of MySQL, as well as the strict SQL mode and how it differs.

If you are not using strict mode, then whenever you insert an “incorrect” value into a column, such as a **NULL** into a **NOT NULL** column or a too-large numeric value into a numeric column, MySQL sets the column to the “best possible value” instead of producing an error: The following rules describe in more detail how this works:

- If you try to store an out of range value into a numeric column, MySQL Server instead stores zero, the smallest possible value, or the largest possible value, whichever is closest to the invalid value.
- For strings, MySQL stores either the empty string or as much of the string as can be stored in the column.
- If you try to store a string that doesn't start with a number into a numeric column, MySQL Server stores 0.
- Invalid values for **ENUM** and **SET** columns are handled as described in [Section 1.9.6.3, “ENUM and SET Constraints”](#).
- MySQL allows you to store certain incorrect date values into **DATE** and **DATETIME** columns (such as `'2000-02-31'` or `'2000-02-00'`). The idea is that it's not the job of the SQL server to validate dates. If MySQL can store a date value and retrieve exactly the same value, MySQL stores it as given. If the date is totally wrong (outside the server's ability to store it), the special “zero” date value `'0000-00-00'` is stored in the column instead.
- If you try to store **NULL** into a column that doesn't take **NULL** values, an error occurs for single-row **INSERT** statements. For multiple-row **INSERT** statements or for **INSERT INTO ... SELECT** statements, MySQL Server stores the implicit default value for the column data type. In general, this is 0 for numeric types, the empty string (`' '`) for string types, and the “zero” value for date and time types. Implicit default values are discussed in [Section 10.1.4, “Data Type Default Values”](#).
- If an **INSERT** statement specifies no value for a column, MySQL inserts its default value if the column definition includes an explicit **DEFAULT** clause. If the definition has no such **DEFAULT** clause, MySQL inserts the implicit default value for the column data type.

The reason for using the preceding rules in non-strict mode is that we can't check these conditions until the statement has begun executing. We can't just roll back if we encounter a problem after updating a few rows, because the storage engine may not support rollback. The option of terminating the statement is not that good; in this case, the update would be “half done,” which is probably the worst possible scenario. In this case, it's better to “do the best you can” and then continue as if nothing happened.

In MySQL 5.0.2 and up, you can select stricter treatment of input values by using the [STRICT_TRANS_TABLES](#) or [STRICT_ALL_TABLES](#) SQL modes:

```
SET sql_mode = 'STRICT_TRANS_TABLES';  
SET sql_mode = 'STRICT_ALL_TABLES';
```

[STRICT_TRANS_TABLES](#) enables strict mode for transactional storage engines, and also to some extent for non-transactional engines. It works like this:

- For transactional storage engines, bad data values occurring anywhere in a statement cause the statement to abort and roll back.
- For non-transactional storage engines, a statement aborts if the error occurs in the first row to be inserted or updated. (When the error occurs in the first row, the statement can be aborted to leave the table unchanged, just as for a transactional table.) Errors in rows after the first do not abort the statement, because the table has already been changed by the first row. Instead, bad data values are adjusted and result in warnings rather than errors. In other words, with [STRICT_TRANS_TABLES](#), a wrong value causes MySQL to roll back all updates done so far, if that can be done without changing the table. But once the table has been changed, further errors result in adjustments and warnings.

For even stricter checking, enable [STRICT_ALL_TABLES](#). This is the same as [STRICT_TRANS_TABLES](#) except that for non-transactional storage engines, errors abort the statement even for bad data in rows following the first row. This means that if an error occurs partway through a multiple-row insert or update for a non-transactional table, a partial update results. Earlier rows are inserted or updated, but those from the point of the error on are not. To avoid this for non-transactional tables, either use single-row statements or else use [STRICT_TRANS_TABLES](#) if conversion warnings rather than errors are acceptable. To avoid problems in the first place, do not use MySQL to check column content. It is safest (and often faster) to let the application ensure that it passes only legal values to the database.

With either of the strict mode options, you can cause errors to be treated as warnings by using [INSERT IGNORE](#) or [UPDATE IGNORE](#) rather than [INSERT](#) or [UPDATE](#) without [IGNORE](#).

1.9.6.3. ENUM and SET Constraints

[ENUM](#) and [SET](#) columns provide an efficient way to define columns that can contain only a given set of values. See [Section 10.4.4, “The ENUM Type”](#), and [Section 10.4.5, “The SET Type”](#). However, before MySQL 5.0.2, [ENUM](#) and [SET](#) columns do not provide true constraints on entry of invalid data:

- [ENUM](#) columns always have a default value. If you specify no default value, then it is [NULL](#) for columns that can have [NULL](#), otherwise it is the first enumeration value in the column definition.
- If you insert an incorrect value into an [ENUM](#) column or if you force a value into an [ENUM](#) column with [IGNORE](#), it is set to the reserved enumeration value of 0, which is displayed as an empty string in string context.
- If you insert an incorrect value into a [SET](#) column, the incorrect value is ignored. For example, if the column can contain the values 'a', 'b', and 'c', an attempt to assign 'a,x,b,y' results in a value of 'a,b'.

As of MySQL 5.0.2, you can configure the server to use strict SQL mode. See [Section 5.2.6, “SQL Modes”](#). With strict mode enabled, the definition of a [ENUM](#) or [SET](#) column does act as a constraint on values entered into the column. An error occurs for values that do not satisfy these conditions:

- An [ENUM](#) value must be one of those listed in the column definition, or the internal numeric equivalent thereof. The value cannot be the error value (that is, 0 or the empty string). For a column defined as [ENUM\('a', 'b', 'c'\)](#), values such as '', 'd', or 'ax' are illegal and are rejected.
- A [SET](#) value must be the empty string or a value consisting only of the values listed in the column definition separated by commas. For a column defined as [SET\('a', 'b', 'c'\)](#), values such as 'd' or 'a,b,c,d' are illegal and are rejected.

Errors for invalid values can be suppressed in strict mode if you use `INSERT IGNORE` or `UPDATE IGNORE`. In this case, a warning is generated rather than an error. For `ENUM`, the value is inserted as the error member (`0`). For `SET`, the value is inserted as given except that any invalid substrings are deleted. For example, `'a, x, b, y'` results in a value of `'a, b'`.

Chapter 2. Installing and Upgrading MySQL

This chapter describes how to obtain and install MySQL. A summary of the procedure follows and later sections provide the details. If you plan to upgrade an existing version of MySQL to a newer version rather than install MySQL for the first time, see [Section 2.11, “Upgrading MySQL”](#), for information about upgrade procedures and about issues that you should consider before upgrading.

If you are interested in migrating to MySQL from another database system, you may wish to read [Section A.8, “MySQL 5.1 FAQ — Migration”](#), which contains answers to some common questions concerning migration issues.

1. **Determine whether MySQL runs and is supported on your platform.** Please note that not all platforms are equally suitable for running MySQL, and that not all platforms on which MySQL is known to run are officially supported by MySQL AB:
 - For MySQL Enterprise Server, the officially supported platforms are listed at <http://www.mysql.com/support/supportedplatforms.html>.
 - MySQL Community Server runs on the platforms listed at [Section 2.1.1, “Operating Systems Supported by MySQL Community Server”](#).
2. **Choose which distribution to install.** Several versions of MySQL are available, and most are available in several distribution formats. You can choose from pre-packaged distributions containing binary (precompiled) programs or source code. When in doubt, use a binary distribution. We also provide public access to our current source tree for those who want to see our most recent developments and help us test new code. To determine which version and type of distribution you should use, see [Section 2.1.2, “Choosing Which MySQL Distribution to Install”](#).
3. **Download the distribution that you want to install.** For instructions, see [Section 2.1.3, “How to Get MySQL”](#). To verify the integrity of the distribution, use the instructions in [Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”](#).
4. **Install the distribution.** To install MySQL from a binary distribution, use the instructions in [Section 2.2, “Standard MySQL Installation Using a Binary Distribution”](#). To install MySQL from a source distribution or from the current development source tree, use the instructions in [Section 2.9, “MySQL Installation Using a Source Distribution”](#).

If you encounter installation difficulties, see [Section 2.13, “Operating System-Specific Notes”](#), for information on solving problems for particular platforms.
5. **Perform any necessary post-installation setup.** After installing MySQL, read [Section 2.10, “Post-Installation Setup and Testing”](#). This section contains important information about making sure the MySQL server is working properly. It also describes how to secure the initial MySQL user accounts, *which have no passwords* until you assign passwords. The section applies whether you install MySQL using a binary or source distribution.
6. If you want to run the MySQL benchmark scripts, Perl support for MySQL must be available. See [Section 2.15, “Perl Installation Notes”](#).

2.1. General Installation Issues

The MySQL installation procedure depends on whether you will install MySQL Enterprise Server or MySQL Community Server. The set of applicable platforms depends on which distribution you will install:

- For MySQL Enterprise Server, the officially supported platforms are listed at <http://www.mysql.com/support/supportedplatforms.html>.
- MySQL Community Server runs on the platforms listed at [Section 2.1.1, “Operating Systems Supported by MySQL Community Server”](#).

For MySQL Enterprise Server, install the main distribution plus any service packs or hotfixes that you wish to apply using the Enterprise Installer. For platforms that do not yet have an Enterprise Installer, use the Community Server instructions.

For MySQL Community Server, install the main distribution plus any hotfixes and updates:

- Download a binary release, or download a source release and build MySQL yourself from the source code.

- Retrieve MySQL from the BitKeeper tree and build it from source. The BitKeeper tree contains the latest developer code.

The immediately following sections contain the information necessary to choose, download, and verify your distribution. The instructions in later sections of the chapter describe how to install the distribution that you choose. For binary distributions, see the instructions at [Section 2.2, “Standard MySQL Installation Using a Binary Distribution”](#). To build MySQL from source, use the instructions at [Section 2.9, “MySQL Installation Using a Source Distribution”](#).

2.1.1. Operating Systems Supported by MySQL Community Server

This section lists the operating systems on which MySQL Community Server is known to run.

Important

MySQL AB does not necessarily provide official support for all the platforms listed in this section. For information about those platforms which MySQL AB officially supports, see [MySQL Server Supported Platforms](#) on the MySQL Web site.

We use GNU Autoconf, so it is possible to port MySQL to all modern systems that have a C++ compiler and a working implementation of POSIX threads. (Thread support is needed for the server. To compile only the client code, the only requirement is a C++ compiler.)

MySQL has been reported to compile successfully on the following combinations of operating system and thread package.

- AIX 4.x, 5.x with native threads. See [Section 2.13.5.3, “IBM-AIX notes”](#).
- Amiga.
- FreeBSD 5.x and up with native threads.
- HP-UX 11.x with the native threads. See [Section 2.13.5.2, “HP-UX Version 11.x Notes”](#).
- Linux, builds on all fairly recent Linux distributions with [glibc](#) 2.3. See [Section 2.13.1, “Linux Notes”](#).
- Mac OS X. See [Section 2.13.2, “Mac OS X Notes”](#).
- NetBSD 1.3/1.4 Intel and NetBSD 1.3 Alpha. See [Section 2.13.4.2, “NetBSD Notes”](#).
- Novell NetWare 6.0 and 6.5. See [Section 2.7, “Installing MySQL on NetWare”](#).
- OpenBSD 2.5 and with native threads. OpenBSD earlier than 2.5 with the MIT-pthreads package. See [Section 2.13.4.3, “OpenBSD 2.5 Notes”](#).
- SCO OpenServer 5.0.X with a recent port of the FSU Pthreads package. See [Section 2.13.5.8, “SCO UNIX and OpenServer 5.0.x Notes”](#).
- SCO Openserver 6.0.x. See [Section 2.13.5.9, “SCO OpenServer 6.0.x Notes”](#).
- SCO UnixWare 7.1.x. See [Section 2.13.5.10, “SCO UnixWare 7.1.x and OpenUNIX 8.0.0 Notes”](#).
- SGI Irix 6.x with native threads. See [Section 2.13.5.7, “SGI Irix Notes”](#).
- Solaris 2.5 and above with native threads on SPARC and x86. See [Section 2.13.3, “Solaris Notes”](#).
- Tru64 Unix. See [Section 2.13.5.5, “Alpha-DEC-UNIX Notes \(Tru64\)”](#).
- Windows 2000, XP, and Windows Server 2003. See [Section 2.3, “Installing MySQL on Windows”](#).

MySQL has also been known to run on other systems in the past. See [Section 2.13, “Operating System-Specific Notes”](#). Some porting effort might be required for current versions of MySQL on these systems.

Not all platforms are equally well-suited for running MySQL. How well a certain platform is suited for a high-load mission-critical MySQL server is determined by the following factors:

- General stability of the thread library. A platform may have an excellent reputation otherwise, but MySQL is only as stable as the

thread library it calls, even if everything else is perfect.

- The capability of the kernel and the thread library to take advantage of symmetric multi-processor (SMP) systems. In other words, when a process creates a thread, it should be possible for that thread to run on a CPU different from the original process.
- The capability of the kernel and the thread library to run many threads that acquire and release a mutex over a short critical region frequently without excessive context switches. If the implementation of `pthread_mutex_lock()` is too anxious to yield CPU time, this hurts MySQL tremendously. If this issue is not taken care of, adding extra CPUs actually makes MySQL slower.
- General filesystem stability and performance.
- If your tables are large, performance is affected by the ability of the filesystem to deal with large files at all and to deal with them efficiently.
- Our level of expertise here at MySQL AB with the platform. If we know a platform well, we enable platform-specific optimizations and fixes at compile time. We can also provide advice on configuring your system optimally for MySQL.
- The amount of testing we have done internally for similar configurations.
- The number of users that have run MySQL successfully on the platform in similar configurations. If this number is high, the likelihood of encountering platform-specific surprises is much smaller.

2.1.2. Choosing Which MySQL Distribution to Install

When preparing to install MySQL, you should decide which version to use. MySQL development occurs in several release series, and you can pick the one that best fits your needs. After deciding which version to install, you can choose a distribution format. Releases are available in binary or source format.

2.1.2.1. Choosing Which Version of MySQL to Install

The first decision to make is whether you want to use a production (stable) release or a development release. In the MySQL development process, multiple release series co-exist, each at a different stage of maturity:

- MySQL 5.1 is the current development release series.
- MySQL 5.0 is the current stable (production-quality) release series. New releases are issued for bugfixes only; no new features are being added that could effect stability.
- MySQL 4.1 is the previous stable (production-quality) release series. New releases are issued for critical bugfixes and security fixes. No significant new features are to be added to this series.
- MySQL 4.0 and 3.23 are the old stable (production-quality) release series. These versions are now retired, so new releases are issued only to fix extremely critical bugs (primarily security issues).

We do not believe in a complete code freeze because this prevents us from making bugfixes and other fixes that must be done. By “somewhat frozen” we mean that we may add small things that should not affect anything that currently works in a production release. Naturally, relevant bugfixes from an earlier series propagate to later series.

Normally, if you are beginning to use MySQL for the first time or trying to port it to some system for which there is no binary distribution, we recommend going with the production release series. Currently, this is MySQL 5.0. All MySQL releases, even those from development series, are checked with the MySQL benchmarks and an extensive test suite before being issued.

If you are running an older system and want to upgrade, but do not want to take the chance of having a non-seamless upgrade, you should upgrade to the latest version in the same release series you are using (where only the last part of the version number is newer than yours). We have tried to fix only fatal bugs and make only small, relatively “safe” changes to that version.

If you want to use new features not present in the production release series, you can use a version from a development series. Note that development releases are not as stable as production releases.

If you want to use the very latest sources containing all current patches and bugfixes, you can use one of our BitKeeper repositories. These are not “releases” as such, but are available as previews of the code on which future releases are to be based.

The MySQL naming scheme uses release names that consist of three numbers and a suffix; for example, **mysql-5.0.12-beta**. The numbers within the release name are interpreted as follows:

- The first number (**5**) is the major version and describes the file format. All MySQL 5 releases have the same file format.
- The second number (**0**) is the release level. Taken together, the major version and release level constitute the release series number.
- The third number (**12**) is the version number within the release series. This is incremented for each new release. Usually you want the latest version for the series you have chosen.

For each minor update, the last number in the version string is incremented. When there are major new features or minor incompatibilities with previous versions, the second number in the version string is incremented. When the file format changes, the first number is increased.

Release names also include a suffix to indicate the stability level of the release. Releases within a series progress through a set of suffixes to indicate how the stability level improves. The possible suffixes are:

- **alpha** indicates that the release is for preview purposes only. Known bugs should be documented in the News section (see [Appendix C, MySQL Change History](#)). Most alpha releases implement new commands and extensions. Active development that may involve major code changes can occur in an alpha release. However, we do conduct testing before issuing a release.

- **beta** indicates that the release is appropriate for use with new development. Within beta releases, the features and compatibility should remain consistent. However, beta releases may contain numerous and major unaddressed bugs.

All APIs, externally visible structures, and columns for SQL statements will not change during future beta, release candidate, or production releases.

- **rc** indicates a Release Candidate. Release candidates are believed to be stable, having passed all of MySQL's internal testing, and with all known fatal runtime bugs fixed. However, the release has not been in widespread use long enough to know for sure that all bugs have been identified. Only minor fixes are added. (A release candidate is what formerly was known as a gamma release.)
- If there is no suffix, it indicates that the release is a General Availability (GA) or Production release. GA releases are stable, having successfully passed through all earlier release stages and are believed to be reliable, free of serious bugs, and suitable for use in production systems. Only critical bugfixes are applied to the release.

MySQL uses a naming scheme that is slightly different from most other products. In general, it is usually safe to use any version that has been out for a couple of weeks without being replaced by a new version within the same release series.

All releases of MySQL are run through our standard tests and benchmarks to ensure that they are relatively safe to use. Because the standard tests are extended over time to check for all previously found bugs, the test suite keeps getting better.

All releases have been tested at least with these tools:

- An internal test suite

The `mysql-test` directory contains an extensive set of test cases. We run these tests for every server binary. See [Section 28.1.2, “MySQL Test Suite”](#), for more information about this test suite.

- The MySQL benchmark suite

This suite runs a range of common queries. It is also a test to determine whether the latest batch of optimizations actually made the code faster. See [Section 6.1.4, “The MySQL Benchmark Suite”](#).

- The `crash-me` test

This test tries to determine what features the database supports and what its capabilities and limitations are. See [Section 6.1.4, “The MySQL Benchmark Suite”](#).

We also test the newest MySQL version in our internal production environment, on at least one machine. We have more than 100GB of data to work with.

2.1.2.2. Choosing a Distribution Format

After choosing which version of MySQL to install, you should decide whether to use a binary distribution or a source distribution. In most cases, you should probably use a binary distribution, if one exists for your platform. Binary distributions are available in native format for many platforms, such as RPM files for Linux or PKG package installers for Mac OS X or Solaris. Distributions also are available as Zip archives or compressed [tar](#) files.

Reasons to choose a binary distribution include the following:

- Binary distributions generally are easier to install than source distributions.
- To satisfy different user requirements, we provide several servers in binary distributions. [mysqld](#) is an optimized server that is a smaller, faster binary. [mysqld-debug](#) is compiled with debugging support.

Each of these servers is compiled from the same source distribution, though with different configuration options. All native MySQL clients can connect to servers from either MySQL version.

Under some circumstances, you may be better off installing MySQL from a source distribution:

- You want to install MySQL at some explicit location. The standard binary distributions are ready to run at any installation location, but you might require even more flexibility to place MySQL components where you want.
- You want to configure [mysqld](#) to ensure that features are available that might not be included in the standard binary distributions. Here is a list of the most common extra options that you may want to use to ensure feature availability:
 - [--with-libwrap](#)
 - [--with-named-z-libs](#) (this is done for some of the binaries)
 - [--with-debug\[=full\]](#)
- You want to configure [mysqld](#) without some features that are included in the standard binary distributions. For example, distributions normally are compiled with support for all character sets. If you want a smaller MySQL server, you can recompile it with support for only the character sets you need.
- You have a special compiler (such as [pgcc](#)) or want to use compiler options that are better optimized for your processor. Binary distributions are compiled with options that should work on a variety of processors from the same processor family.
- You want to use the latest sources from one of the BitKeeper repositories to have access to all current bugfixes. For example, if you have found a bug and reported it to the MySQL development team, the bugfix is committed to the source repository and you can access it there. The bugfix does not appear in a release until a release actually is issued.
- You want to read (or modify) the C and C++ code that makes up MySQL. For this purpose, you should get a source distribution, because the source code is always the ultimate manual.
- Source distributions contain more tests and examples than binary distributions.

2.1.2.3. How and When Updates Are Released

MySQL is evolving quite rapidly and we want to share new developments with other MySQL users. We try to produce a new release whenever we have new and useful features that others also seem to have a need for.

We also try to help users who request features that are easy to implement. We take note of what our licensed users want, and we especially take note of what our support customers want and try to help them in this regard.

No one is *required* to download a new release. The News section helps you determine whether the new release has something you really want. See [Appendix C, MySQL Change History](#).

We use the following policy when updating MySQL:

- Enterprise Server releases are meant to appear every 18 months, supplemented by quarterly service packs and monthly rapid up-

dates. Community Server releases are meant to appear 2–3 times per year.

- Releases are issued within each series. Enterprise Server releases are numbered using even numbers (for example, 5.1.20). Community Server releases are numbered using odd numbers (for example, 5.1.21).
- Binary distributions for some platforms are made by us for major releases. Other people may make binary distributions for other systems, but probably less frequently.
- We make fixes available as soon as we have identified and corrected small or non-critical but annoying bugs. The fixes are available in source form immediately from our public BitKeeper repositories, and are included in the next release.
- If by any chance a security vulnerability or critical bug is found in a release, our policy is to fix it in a new release as soon as possible. (We would like other companies to do this, too!)

2.1.2.4. Release Philosophy—No Known Bugs in Releases

We put considerable time and effort into making our releases bug-free. Our policy is never to release a version of MySQL intended for production use that has any known fatal, repeatable bugs.

We have documented all open problems, bugs, and issues that are dependent on design decisions. See [Section B.1.8, “Known Issues in MySQL”](#).

Our aim is to fix everything that is fixable without making a stable MySQL version less stable. In certain cases, this means we can fix an issue in the development versions, but not in the stable (production) version. Naturally, we document such issues so that users are aware of them.

Here is a description of our build process:

- We monitor bugs from our customer support list, the bugs database at <http://bugs.mysql.com/>, and the MySQL external mailing lists.
- All reported bugs for live versions are entered into the bugs database.
- When we fix a bug, we always try to make a test case for it and include it into our test system to ensure that the bug can never recur without being detected. (About 90% of all fixed bugs have test cases.)
- We create test cases for each new feature that we add to MySQL.
- Before we start to build a new MySQL release, we ensure that all reported repeatable bugs for that MySQL version (3.23.x, 4.0.x, 4.1.x, 5.0.x, 5.1.x, and so on) are fixed. If something is impossible to fix due to some internal design decision in MySQL, we document this in the manual. See [Section B.1.8, “Known Issues in MySQL”](#).
- We do a build on all platforms for which we support binaries and run our test suite and benchmark suite on all of them.
- We do not publish a binary for a platform for which the test or benchmark suite fails. If the problem is due to a general error in the source, we fix it and do the build plus tests on all systems again from scratch.
- The build and test process takes a week. If we receive a report regarding a fatal bug during this process (for example, one that causes a core dump), we fix the problem and restart the build process.
- After publishing the binaries on <http://dev.mysql.com/>, we send out an announcement message to the [mysql](#) and [announce](#) mailing lists. See [Section 1.7.1, “MySQL Mailing Lists”](#). The announcement message contains a list of all changes to the release and any known problems with the release. The **Known Problems** section in the release notes has been needed for only a handful of releases.
- To quickly give our users access to the latest MySQL features, we try to produce a new MySQL release every 4-8 weeks. Source code snapshots are built daily and are available at <http://downloads.mysql.com/snapshots.php>.
- If, despite our best efforts, we receive any bug reports after a release is issued that a critical problem exists for the build on a specific platform, we fix it at once and build a new 'a' release for that platform. Thanks to our large user base, problems are found and resolved very quickly.
- Our track record for making stable releases is quite good. In the last 150 releases, we had to do a new build for fewer than 10 of them. In three of these cases, the bug was a faulty [glibc](#) library on one of our build machines that took us a long time to track down.

2.1.2.5. MySQL Binaries Compiled by MySQL AB

As a service of MySQL AB, we provide a set of binary distributions of MySQL that are compiled on systems at our site or on systems where supporters of MySQL kindly have given us access to their machines.

In addition to the binaries provided in platform-specific package formats, we offer binary distributions for a number of platforms in the form of compressed **tar** files (**.tar.gz** files). See [Section 2.2, “Standard MySQL Installation Using a Binary Distribution”](#).

The RPM distributions for MySQL 5.1 releases that we make available through our Web site are generated by MySQL AB.

For Windows distributions, see [Section 2.3, “Installing MySQL on Windows”](#).

These distributions are generated using the script `Build-tools/Do-compile`, which compiles the source code and creates the binary **tar.gz** archive using `scripts/make_binary_distribution`.

These binaries are configured and built with the following compilers and options. This information can also be obtained by looking at the variables `COMP_ENV_INFO` and `CONFIGURE_LINE` inside the script `bin/mysqlbug` of every binary **tar** file distribution.

Anyone who has more optimal options for any of the following `configure` commands can mail them to the MySQL [internals](#) mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#).

If you want to compile a debug version of MySQL, you should add `--with-debug` or `--with-debug=full` to the following `configure` commands and remove any `-fomit-frame-pointer` options.

The following binaries are built on MySQL AB development systems:

- Linux 2.4.xx x86 with **gcc** 2.95.3:

```
CFLAGS="-O2 -mcpu=pentiumpro" CXX=gcc CXXFLAGS="-O2 -mcpu=pentiumpro
-felide-constructors" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-asm --disable-shared
--with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static
```

- Linux 2.4.x x86 with **icc** (Intel C++ Compiler 8.1 or later releases):

```
CC=icc CXX=icpc CFLAGS="-O3 -unroll2 -ip -mp -no-gcc -restrict"
CXXFLAGS="-O3 -unroll2 -ip -mp -no-gcc -restrict" ./configure
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --enable-asm
--disable-shared --with-client-ldflags=-all-static
--with-mysqld-ldflags=-all-static --with-embedded-server --with-innodb
```

Note that versions 8.1 and newer of the Intel compiler have separate drivers for 'pure' C (**icc**) and C++ (**icpc**); if you use **icc** version 8.0 or older for building MySQL, you will need to set `CXX=icc`.

- Linux 2.4.xx Intel Itanium 2 with **ecc** (Intel C++ Itanium Compiler 7.0):

```
CC=ecc CFLAGS="-O2 -tpp2 -ip -nolib_inline" CXX=ecc CXXFLAGS="-O2
-tpp2 -ip -nolib_inline" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile
```

- Linux 2.4.xx Intel Itanium with **ecc** (Intel C++ Itanium Compiler 7.0):

```
CC=ecc CFLAGS=-tpp1 CXX=ecc CXXFLAGS=-tpp1 ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile
```

- Linux 2.4.xx alpha with **ccc** (Compaq C V6.2-505 / Compaq C++ V6.3-006):

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx CXXFLAGS="-fast -arch
generic -noexceptions -nortti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-mysqld-ldflags=-non_shared
--with-client-ldflags=-non_shared --disable-shared
```


- Linux 2.x.xx ppc with [gcc 2.95.4](#):

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared --with-embedded-server
--with-innodb
```

- Linux 2.4.xx s390 with [gcc 2.95.3](#):

```
CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors" ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
--with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static
```

- Linux 2.4.xx x86_64 (AMD64) with [gcc 3.2.1](#):

```
CXX=gcc ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared
```

- Sun Solaris 8 x86 with [gcc 3.2.3](#):

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared --with-innodb
```

- Sun Solaris 8 SPARC with [gcc 3.2](#):

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-assembler --with-named-z-libs=no
--with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 8 SPARC 64-bit with [gcc 3.2](#):

```
CC=gcc CFLAGS="-O3 -m64 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-m64 -fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no
--with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 9 SPARC with [gcc 2.95.3](#):

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-assembler --with-named-curses-libs=-lcurses
--disable-shared
```

- Sun Solaris 9 SPARC with [cc-5.0](#) (Sun Forte 5.0):

```
CC=cc-5.0 CXX=CC ASFLAGS="-xarch=v9" CFLAGS="-Xa -xstrconst -mt
-D_FORTERC_ -xarch=v9" CXXFLAGS="-noex -mt -D_FORTERC_ -xarch=v9"
./configure --prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --enable-assembler
--with-named-z-libs=no --enable-thread-safe-client --disable-shared
```

- IBM AIX 4.3.2 ppc with [gcc 3.2.3](#):

```
CFLAGS="-O2 -mcpu=powerpc -Wa,-many " CXX=gcc CXXFLAGS="-O2
-mcpu=powerpc -Wa,-many -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
```

```
--enable-local-infile --with-named-z-libs=no --disable-shared
```

- IBM AIX 4.3.3 ppc with **xlc_r** (IBM Visual Age C/C++ 6.0):

```
CC=xlc_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
CXX=xlc_r CXXFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --with-named-z-libs=no
--disable-shared --with-innodb
```

- IBM AIX 5.1.0 ppc with **gcc** 3.3:

```
CFLAGS="-O2 -mcpu=powerpc -Wa,-many" CXX=gcc CXXFLAGS="-O2 -mcpu=powerpc
-Wa,-many -felide-constructors -fno-exceptions -fno-rtti" ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --with-named-z-libs=no
--disable-shared
```

- IBM AIX 5.2.0 ppc with **xlc_r** (IBM Visual Age C/C++ 6.0):

```
CC=xlc_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
CXX=xlc_r CXXFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --with-named-z-libs=no
--disable-shared --with-embedded-server --with-innodb
```

- HP-UX 10.20 pa-risc1.1 with **gcc** 3.1:

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc CXXFLAGS="-DHPUX
-I/opt/dce/include -felide-constructors -fno-exceptions -fno-rtti
-O3 -fPIC" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-pthread --with-named-thread-libs=-ldce
--with-lib-ccflags=-fPIC --disable-shared
```

- HP-UX 11.00 pa-risc with **aCC** (HP ANSI C++ B3910B A.03.50):

```
CC=cc CXX=aCC CFLAGS="+DAportable" CXXFLAGS="+DAportable" ./configure
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
--with-embedded-server --with-innodb
```

- HP-UX 11.11 pa-risc2.0 64bit with **aCC** (HP ANSI C++ B3910B A.03.33):

```
CC=cc CXX=aCC CFLAGS="+DD64" CXXFLAGS="+DD64" ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
```

- HP-UX 11.11 pa-risc2.0 32bit with **aCC** (HP ANSI C++ B3910B A.03.33):

```
CC=cc CXX=aCC CFLAGS="+DAportable" CXXFLAGS="+DAportable" ./configure
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
--with-innodb
```

- HP-UX 11.22 ia64 64bit with **aCC** (HP aC++/ANSI C B3910B A.05.50):

```
CC=cc CXX=aCC CFLAGS="+DD64 +DSitanium2" CXXFLAGS="+DD64 +DSitanium2"
./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
--with-embedded-server --with-innodb
```

- Apple Mac OS X 10.2 powerpc with **gcc** 3.1:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
```

```
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared
```

- FreeBSD 4.7 i386 with [gcc](#) 2.95.4:

```
CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-assembler --with-named-z-libs=not-used
--disable-shared
```

- FreeBSD 4.7 i386 using LinuxThreads with [gcc](#) 2.95.4:

```
CFLAGS="-DHAVE_BROKEN_REALPATH -D__USE_UNIX98 -D_REENTRANT
-D_THREAD_SAFE -I/usr/local/include/pthread/linuxthreads"
CXXFLAGS="-DHAVE_BROKEN_REALPATH -D__USE_UNIX98 -D_REENTRANT
-D_THREAD_SAFE -I/usr/local/include/pthread/linuxthreads" ./configure
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --enable-thread-safe-client
--enable-local-infile --enable-assembler
--with-named-thread-libs="-DHAVE_GLIBC2_STYLE_GETHOSTBYNAME_R
-D_THREAD_SAFE -I /usr/local/include/pthread/linuxthreads
-L/usr/local/lib -llthread -llgcc_r" --disable-shared
--with-embedded-server --with-innodb
```

- QNX Neutrino 6.2.1 i386 with [gcc](#) 2.95.3qnx-nto 20010315:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared
```

The following binaries are built on third-party systems kindly provided to MySQL AB by other users. These are provided only as a courtesy; MySQL AB does not have full control over these systems, so we can provide only limited support for the binaries built on them.

- SCO Unix 3.2v5.0.7 i386 with [gcc](#) 2.95.3:

```
CFLAGS="-O3 -mpentium" LDFLAGS=-static CXX=gcc CXXFLAGS="-O3 -mpentium
-felide-constructors" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --enable-thread-safe-client
--disable-shared
```

- SCO UnixWare 7.1.4 i386 with [CC](#) 3.2:

```
CC=cc CFLAGS="-O" CXX=CC ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --enable-thread-safe-client
--disable-shared --with-readline
```

- SCO OpenServer 6.0.0 i386 with [CC](#) 3.2:

```
CC=cc CFLAGS="-O" CXX=CC ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --enable-thread-safe-client
--disable-shared --with-readline
```

- Compaq Tru64 OSF/1 V5.1 732 alpha with [cc/cxx](#) (Compaq C V6.3-029i / DIGITAL C++ V6.1-027):

```
CC="cc -pthread" CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline
speed -speculate all" CXX="cxx -pthread" CXXFLAGS="-O4 -ansi_alias
-fast -inline speed -speculate all -noexceptions -nortti" ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile
--with-named-thread-libs="-lpthread -lmach -lexc -lc" --disable-shared
--with-mysqld-ldflags=-all-static
```

- SGI Irix 6.5 IP32 with [gcc](#) 3.0.1:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXXFLAGS="-O3  
-fno-omit-frame-pointer -felide-constructors -fno-exceptions  
-fno-rtti" ./configure --prefix=/usr/local/mysql  
--with-extra-charsets=complex --enable-thread-safe-client  
--enable-local-infile --disable-shared
```

- FreeBSD/sparc64 5.0 with [gcc](#) 3.2.1:

```
CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/mysql  
--localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin  
--with-extra-charsets=complex --enable-thread-safe-client  
--enable-local-infile --disable-shared --with-innodb
```

The following compile options have been used for binary packages that MySQL AB provided in the past. These binaries no longer are being updated, but the compile options are listed here for reference purposes.

- Linux 2.2.xx SPARC with [egcs](#) 1.1.2:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3  
-fno-omit-frame-pointer -felide-constructors -fno-exceptions  
-fno-rtti" ./configure --prefix=/usr/local/mysql  
--with-extra-charsets=complex --enable-thread-safe-client  
--enable-local-infile --enable-assembler --disable-shared
```

- Linux 2.2.x with x86 with [gcc](#) 2.95.2:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro  
-felide-constructors -fno-exceptions -fno-rtti" ./configure  
--prefix=/usr/local/mysql --enable-assembler  
--with-mysqld-ldflags=-all-static --disable-shared  
--with-extra-charsets=complex
```

- SunOS 4.1.4 2 sun4c with [gcc](#) 2.7.2.1:

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors" ./configure  
--prefix=/usr/local/mysql --disable-shared --with-extra-charsets=complex  
--enable-assembler
```

- SunOS 5.5.1 (and above) sun4u with [egcs](#) 1.0.3a or 2.90.27 or [gcc](#) 2.95.2 and newer:

```
CC=gcc CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors  
-fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql  
--with-low-memory --with-extra-charsets=complex --enable-assembler
```

- SunOS 5.6 i86pc with [gcc](#) 2.8.1:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql  
--with-low-memory --with-extra-charsets=complex
```

- BSDI BSD/OS 3.1 i386 with [gcc](#) 2.7.2.1:

```
CC=gcc CXX=gcc CXXFLAGS=-O ./configure --prefix=/usr/local/mysql  
--with-extra-charsets=complex
```

- BSDI BSD/OS 2.1 i386 with [gcc](#) 2.7.2:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql  
--with-extra-charsets=complex
```

- AIX 4.2 with [gcc](#) 2.7.2.2:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql  
--with-extra-charsets=complex
```

2.1.3. How to Get MySQL

Check our downloads page at <http://dev.mysql.com/downloads/> for information about the current version of MySQL and for downloading instructions. For a complete up-to-date list of MySQL download mirror sites, see <http://dev.mysql.com/downloads/mirrors.html>. You can also find information there about becoming a MySQL mirror site and how to report a bad or out-of-date mirror.

Our main mirror is located at <http://mirrors.sunsite.dk/mysql/>.

2.1.4. Verifying Package Integrity Using MD5 Checksums or GnuPG

After you have downloaded the MySQL package that suits your needs and before you attempt to install it, you should make sure that it is intact and has not been tampered with. MySQL AB offers three means of integrity checking:

- MD5 checksums
- Cryptographic signatures using **GnuPG**, the GNU Privacy Guard
- For RPM packages, the built-in RPM integrity verification mechanism

The following sections describe how to use these methods.

If you notice that the MD5 checksum or GPG signatures do not match, first try to download the respective package one more time, perhaps from another mirror site. If you repeatedly cannot successfully verify the integrity of the package, please notify us about such incidents, including the full package name and the download site you have been using, at [<webmaster@mysql.com>](mailto:webmaster@mysql.com) or [<build@mysql.com>](mailto:build@mysql.com). Do not report downloading problems using the bug-reporting system.

2.1.4.1. Verifying the MD5 Checksum

After you have downloaded a MySQL package, you should make sure that its MD5 checksum matches the one provided on the MySQL download pages. Each package has an individual checksum that you can verify with the following command, where *package_name* is the name of the package you downloaded:

```
shell> md5sum package_name
```

Example:

```
shell> md5sum mysql-standard-5.1.21-beta-linux-i686.tar.gz
aaab65abbec64d5e907dcd41b8699945  mysql-standard-5.1.21-beta-linux-i686.tar.gz
```

You should verify that the resulting checksum (the string of hexadecimal digits) matches the one displayed on the download page immediately below the respective package.

Note

Make sure to verify the checksum of the *archive file* (for example, the *.zip* or *.tar.gz* file) and not of the files that are contained inside of the archive.

Note that not all operating systems support the `md5sum` command. On some, it is simply called `md5`, and others do not ship it at all. On Linux, it is part of the **GNU Text Utilities** package, which is available for a wide range of platforms. You can download the source code from <http://www.gnu.org/software/textutils/> as well. If you have OpenSSL installed, you can use the command `openssl md5 package_name` instead. A Windows implementation of the `md5` command line utility is available from <http://www.fourmilab.ch/md5/>. `winMd5Sum` is a graphical MD5 checking tool that can be obtained from <http://www.nullriver.com/index/products/winmd5sum>.

2.1.4.2. Signature Checking Using GnuPG

Another method of verifying the integrity and authenticity of a package is to use cryptographic signatures. This is more reliable than using MD5 checksums, but requires more work.

At MySQL AB, we sign MySQL downloadable packages with **GnuPG** (GNU Privacy Guard). **GnuPG** is an Open Source alternative to the well-known Pretty Good Privacy (**PGP**) by Phil Zimmermann. See <http://www.gnupg.org/> for more information about **GnuPG** and how to obtain and install it on your system. Most Linux distributions ship with **GnuPG** installed by default. For more information about

GnuPG, see <http://www.openpgp.org/>.

To verify the signature for a specific package, you first need to obtain a copy of MySQL AB's public GPG build key, which you can download from <http://www.keyservers.net/>. The key that you want to obtain is named build@mysql.com. Alternatively, you can cut and paste the key directly from the following text:

```
Key ID:
pub 1024D/5072E1F5 2003-02-03
    MySQL Package signing key (www.mysql.com) <build@mysql.com>
Fingerprint: A4A9 4068 76FC BD3C 4567 70C8 8C71 8D3B 5072 E1F5

Public Key (ASCII-armored):

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org

mQG6iBD4+owwRBAC14GIfuFCyEDSIEpVEW3SAFUdJBtoQHH/nJKZyQT7h9bPLUWC3
RODjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPKbDcK96+OmSLN9brZ
fw2v0UgCmYv2hW0hyDHuvYlQA/BThQoADgJ8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
Bq0xRznNCRRCrAuAuVztHRCEAJooQK1+iSiunZMYD1WufEXfshc57S/+yeJkegnW
hxrR9pRWArNyJdDRT++f2RUE3vpquKNQU/hnEIHJRQqYHo8gTxvxxNQC7fJYLV
K2HtrKpPb72vwsEKMYhhr0eKcBtLGfLs9krjJ6sBgACyP/Vb7hiPwxh6rDZITnE
kYpXBACmWpP8NJTkamEnPCia2Zo0HODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSb10LLtZciNLYsaFAPEOMDKpMqAK6IyisNtPvalD8lH0bPAnWqcyefep
rv0sxxqUEmC3o7wWgfN83P0kDasDbs3pjwPhxvz6//62zQJ7Q7TXLTUuWgUGFj
a2FnxSBzawduaw5nIGtLeSAod3d3Lm15c3FsLmNvbSkgPGJ1awxkQG15c3FsLmNv
bt6IXQQTETQIAHQUcPj6jDAUJCWYBgAULBwoDBAMVAwIDFgIBAheAAAJEixxjTtQ
cuH1cY4AnilUwTxn8MatQ0iG0a/bPxrVKGCAJ4oinSNZRYTnblChwFaazt7PF3q
zIhMBBMRagAMBQI+PqPRBYMJZgC7AAoJEElQ4SqycpHyJ0Ean1mxHijft00bKXvu
Cso/pECUmp1AJ4IM9MRVj5VcdH/KN/KjRtW6tHFPYhMBBMRagAMBQI+QoIDBYMJ
YiKJAAoJELb1zU3GuiQ/lpEAOIhpb6BozKI8p6eaaabzF5MLJH58pAKCu/RoofK8J
Eg2aLos+5zEYrB/LsrkCDQq+PqMdEAgA7+GJfxbMdY4wsLpNjH9rF4N2qfWsEN/L
xaZoJyc3a6M02wCnHl6ahT2/tBK2w1QI4YFteR47gCvtgb601JHff0o2HfLmRDRi
Rjd1DTCheqyX7CHhcgHj/dNRlw2Z0l5QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hkAwZE
7zaD5cH9J7yv/6xuZVw411x0h4UqsTcWmu0iM1BzELqX1DY7LwoPEB/09Rkbf4fm
Le11EzIaCa4PqARXQZc4dhSinMt6K3X4BrRsKTfozBu74F47D8I1bf5vSYHbuE5p
/1oIDnkg/p8kw+3FfuWrycciqFTcNz215yyX39LXFnlLzKUb/F5GwADBQf+LwqQ
a8CGRfRs0AJxiM63CHfTy5mUc5rUSntSLGYEIOCR1BeQauyPZbPDsDD9MZ1ZaSaF
anFvWFG6Llx9xkU7tzq+vKLowkm4u5xf3vn55VjnSd1aQ9eQnUcX1L4cnBGoTbOW
I39ECyzgsLzBdC++MPjCQTCA7p6JUvSP6oAB3FQwg54tuUo0Ec8bsM8b3Ev42Lmu
QT5NdKHGwHsXTptl0klk4bQk40ajHsiy1BMahpT27jWjJlMiJc+IWJ0mghkKHT92
6s/ymfdf5Hkd1cyvzsz5tryVI3Fx78XeSYfQvuwwq2H139pXGEkg0n6Kdu0etdZ
Whe70YGNPw1yJWJT1IhMBGqRagAMBQI+PqMdBQkZGGAaAJEixxjTtQcuH17p4A
n3r1QpVC9yhnW2cSAjq+kr72GX0eAJ4295kl6NxYEuFapmr1+0uUq/SlSQ==
=YJkx
-----END PGP PUBLIC KEY BLOCK-----
```

To import the build key into your personal public GPG keyring, use `gpg --import`. For example, if you have saved the key in a file named `mysql_pubkey.asc`, the import command looks like this:

```
shell> gpg --import mysql_pubkey.asc
```

After you have downloaded and imported the public build key, download your desired MySQL package and the corresponding signature, which also is available from the download page. The signature file has the same name as the distribution file with an `.asc` extension. For example:

Distribution file	<code>mysql-standard-5.1.21-beta-linux-i686.tar.gz</code>
Signature file	<code>mysql-standard-5.1.21-beta-linux-i686.tar.gz.asc</code>

Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file:

```
shell> gpg --verify package_name.asc
```

Example:

```
shell> gpg --verify mysql-standard-5.1.21-beta-linux-i686.tar.gz.asc
gpg: Signature made Tue 12 Jul 2005 23:35:41 EST using DSA key ID 5072E1F5
gpg: Good signature from "MySQL Package signing key (www.mysql.com) <build@mysql.com>"
```

The **Good signature** message indicates that everything is all right. You can ignore any **insecure memory** warning you might obtain.

See the GPG documentation for more information on how to work with public keys.

2.1.4.3. Signature Checking Using RPM

For RPM packages, there is no separate signature. RPM packages have a built-in GPG signature and MD5 checksum. You can verify a package by running the following command:

```
shell> rpm --checksig package_name.rpm
```

Example:

```
shell> rpm --checksig MySQL-server-5.1.21-beta-0.glibc23.i386.rpm
MySQL-server-5.1.21-beta-0.glibc23.i386.rpm: md5 gpg OK
```

Note

If you are using RPM 4.1 and it complains about (GPG) NOT OK (MISSING KEYS: GPG#5072e1f5), even though you have imported the MySQL public build key into your own GPG keyring, you need to import the key into the RPM keyring first. RPM 4.1 no longer uses your personal GPG keyring (or GPG itself). Rather, it maintains its own keyring because it is a system-wide application and a user's GPG public keyring is a user-specific file. To import the MySQL public key into the RPM keyring, first obtain the key as described in [Section 2.1.4.2, “Signature Checking Using GnuPG”](#). Then use `rpm --import` to import the key. For example, if you have saved the public key in a file named `mysql_pubkey.asc`, import it using this command:

```
shell> rpm --import mysql_pubkey.asc
```

If you need to obtain the MySQL public key, see [Section 2.1.4.2, “Signature Checking Using GnuPG”](#).

2.1.5. Installation Layouts

This section describes the default layout of the directories created by installing binary or source distributions provided by MySQL AB. A distribution provided by another vendor might use a layout different from those shown here.

For MySQL 5.1 on Windows, the default installation directory is `C:\Program Files\MySQL\MySQL Server 5.1`. (Some Windows users prefer to install in `C:\mysql`, the directory that formerly was used as the default. However, the layout of the subdirectories remains the same.) The installation directory has the following subdirectories:

Directory	Contents of Directory
<code>bin</code>	Client programs and the <code>mysqld</code> server
<code>data</code>	Log files, databases
<code>Docs</code>	Manual in CHM format
<code>examples</code>	Example programs and scripts
<code>include</code>	Include (header) files
<code>lib</code>	Libraries
<code>scripts</code>	Utility scripts
<code>share</code>	Error message files

Installations created from MySQL AB's Linux RPM distributions result in files under the following system directories:

Directory	Contents of Directory
<code>/usr/bin</code>	Client programs and scripts
<code>/usr/sbin</code>	The <code>mysqld</code> server
<code>/var/lib/mysql</code>	Log files, databases
<code>/usr/share/info</code>	Manual in Info format
<code>/usr/share/man</code>	Unix manual pages
<code>/usr/include/mysql</code>	Include (header) files

/usr/lib/mysql	Libraries
/usr/share/mysql	Error message and character set files
/usr/share/sql-bench	Benchmarks

On Unix, a [tar](#) file binary distribution is installed by unpacking it at the installation location you choose (typically [/usr/local/mysql](#)) and creates the following directories in that location:

Directory	Contents of Directory
bin	Client programs and the mysqld server
data	Log files, databases
docs	Manual in Info format
man	Unix manual pages
include	Include (header) files
lib	Libraries
scripts	mysql_install_db
share/mysql	Error message files
sql-bench	Benchmarks

A source distribution is installed after you configure and compile it. By default, the installation step installs files under [/usr/local](#), in the following subdirectories:

Directory	Contents of Directory
bin	Client programs and scripts
include/mysql	Include (header) files
Docs	Manual in Info, CHM formats
man	Unix manual pages
lib/mysql	Libraries
libexec	The mysqld server
share/mysql	Error message files
sql-bench	Benchmarks and crash-me test
var	Databases and log files

Within its installation directory, the layout of a source installation differs from that of a binary installation in the following ways:

- The [mysqld](#) server is installed in the [libexec](#) directory rather than in the [bin](#) directory.
- The data directory is [var](#) rather than [data](#).
- [mysql_install_db](#) is installed in the [bin](#) directory rather than in the [scripts](#) directory.
- The header file and library directories are [include/mysql](#) and [lib/mysql](#) rather than [include](#) and [lib](#).

You can create your own binary installation from a compiled source distribution by executing the [scripts/make_binary_distribution](#) script from the top directory of the source distribution.

2.2. Standard MySQL Installation Using a Binary Distribution

The next several sections cover the installation of MySQL on platforms where we offer packages using the native packaging format of the respective platform. (This is also known as performing a “binary install.”) However, binary distributions of MySQL are available for many other platforms as well. See [Section 2.8, “Installing MySQL from tar .gz Packages on Other Unix-Like Systems”](#), for generic

installation instructions for these packages that apply to all platforms.

See [Section 2.1, “General Installation Issues”](#), for more information on what other binary distributions are available and how to obtain them.

2.3. Installing MySQL on Windows

A native Windows distribution of MySQL has been available from MySQL AB since version 3.21 and represents a sizable percentage of the daily downloads of MySQL. This section describes the process for installing MySQL on Windows.

Note

If you are upgrading MySQL from an existing installation older than MySQL 4.1.5, you must first perform the procedure described in [Section 2.3.14, “Upgrading MySQL on Windows”](#).

To run MySQL on Windows, you need the following:

- A 32-bit Windows operating system such as 2000, XP, Vista, or Windows Server 2003.

A Windows operating system permits you to run the MySQL server as a service. See [Section 2.3.11, “Starting MySQL as a Windows Service”](#).

Generally, you should install MySQL on Windows using an account that has administrator rights. Otherwise, you may encounter problems with certain operations such as editing the `PATH` environment variable or accessing the [Service Control Manager](#).

- TCP/IP protocol support.
- Enough space on the hard drive to unpack, install, and create the databases in accordance with your requirements (generally a minimum of 200 megabytes is recommended.)

There may also be other requirements, depending on how you plan to use MySQL:

- If you plan to connect to the MySQL server via ODBC, you need a Connector/ODBC driver. See [Chapter 26, Connectors](#).
- If you need tables with a size larger than 4GB, install MySQL on an NTFS or newer filesystem. Don't forget to use `MAX_ROWS` and `AVG_ROW_LENGTH` when you create tables. See [Section 12.1.10, “CREATE TABLE Syntax”](#).

MySQL for Windows is available in several distribution formats:

- Binary distributions are available that contain a setup program that installs everything you need so that you can start the server immediately. Another binary distribution format contains an archive that you simply unpack in the installation location and then configure yourself. For details, see [Section 2.3.1, “Choosing An Installation Package”](#).
- The source distribution contains all the code and support files for building the executables using the Visual Studio compiler system.

Generally speaking, you should use a binary distribution that includes an installer. It is simpler to use than the others, and you need no additional tools to get MySQL up and running. The installer for the Windows version of MySQL, combined with a GUI Configuration Wizard, automatically installs MySQL, creates an option file, starts the server, and secures the default user accounts.

The following section describes how to install MySQL on Windows using a binary distribution. To use an installation package that does not include an installer, follow the procedure described in [Section 2.3.5, “Installing MySQL from a Noinstall Zip Archive”](#). To install using a source distribution, see [Section 2.9.6, “Installing MySQL from Source on Windows”](#).

MySQL distributions for Windows can be downloaded from <http://dev.mysql.com/downloads/>. See [Section 2.1.3, “How to Get MySQL”](#).

2.3.1. Choosing An Installation Package

For MySQL 5.1, there are three installation packages to choose from when installing MySQL on Windows:

- **The Essentials Package:** This package has a filename similar to [mysql-essential-5.1.21-beta-win32.msi](#) and contains the minimum set of files needed to install MySQL on Windows, including the Configuration Wizard. This package does not include optional components such as the embedded server and benchmark suite.
- **The Complete Package:** This package has a filename similar to [mysql-5.1.21-beta-win32.zip](#) and contains all files needed for a complete Windows installation, including the Configuration Wizard. This package includes optional components such as the embedded server and benchmark suite.
- **The Noinstall Archive:** This package has a filename similar to [mysql-noinstall-5.1.21-beta-win32.zip](#) and contains all the files found in the Complete install package, with the exception of the Configuration Wizard. This package does not include an automated installer, and must be manually installed and configured.

The Essentials package is recommended for most users. It is provided as an [.msi](#) file for use with the Windows Installer. The Complete and Noinstall distributions are packaged as Zip archives. To use them, you must have a tool that can unpack [.zip](#) files.

Your choice of install package affects the installation process you must follow. If you choose to install either the Essentials or Complete install packages, see [Section 2.3.2, “Installing MySQL with the Automated Installer”](#). If you choose to install MySQL from the Noinstall archive, see [Section 2.3.5, “Installing MySQL from a Noinstall Zip Archive”](#).

2.3.2. Installing MySQL with the Automated Installer

New MySQL users can use the MySQL Installation Wizard and MySQL Configuration Wizard to install MySQL on Windows. These are designed to install and configure MySQL in such a way that new users can immediately get started using MySQL.

The MySQL Installation Wizard and MySQL Configuration Wizard are available in the Essentials and Complete install packages. They are recommended for most standard MySQL installations. Exceptions include users who need to install multiple instances of MySQL on a single server host and advanced users who want complete control of server configuration.

2.3.3. Using the MySQL Installation Wizard

2.3.3.1. Introduction to the Installation Wizard

MySQL Installation Wizard is an installer for the MySQL server that uses the latest installer technologies for Microsoft Windows. The MySQL Installation Wizard, in combination with the MySQL Configuration Wizard, allows a user to install and configure a MySQL server that is ready for use immediately after installation.

The MySQL Installation Wizard is the standard installer for all MySQL server distributions, version 4.1.5 and higher. Users of previous versions of MySQL need to shut down and remove their existing MySQL installations manually before installing MySQL with the MySQL Installation Wizard. See [Section 2.3.3.7, “Upgrading MySQL with the Installation Wizard”](#), for more information on upgrading from a previous version.

Microsoft has included an improved version of their Microsoft Windows Installer (MSI) in the recent versions of Windows. MSI has become the de-facto standard for application installations on Windows 2000, Windows XP, and Windows Server 2003. The MySQL Installation Wizard makes use of this technology to provide a smoother and more flexible installation process.

The Microsoft Windows Installer Engine was updated with the release of Windows XP; those using a previous version of Windows can reference [this Microsoft Knowledge Base article](#) for information on upgrading to the latest version of the Windows Installer Engine.

In addition, Microsoft has introduced the WiX (Windows Installer XML) toolkit recently. This is the first highly acknowledged Open Source project from Microsoft. We have switched to WiX because it is an Open Source project and it allows us to handle the complete Windows installation process in a flexible manner using scripts.

Improving the MySQL Installation Wizard depends on the support and feedback of users like you. If you find that the MySQL Installation Wizard is lacking some feature important to you, or if you discover a bug, please report it in our bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#).

2.3.3.2. Downloading and Starting the MySQL Installation Wizard

The MySQL installation packages can be downloaded from <http://dev.mysql.com/downloads/>. If the package you download is contained within a Zip archive, you need to extract the archive first.

Note

If you are installing on Windows Vista it is best to open a port before beginning the installation. To do this first ensure that you are logged in as an administrator, go to the [Control Panel](#), and double click the [Windows Firewall](#) icon. Choose the [Allow a program through Windows Firewall](#) option and click the ADD PORT button. Enter [MySQL](#) into the **NAME** text box and [3306](#) (or the port of your choice) into the **PORT NUMBER** text box. Also ensure that the **TCP** protocol radio button is selected. If you wish, you can also limit access to the MySQL server by choosing the **CHANGE SCOPE** button. Confirm your choices by clicking the OK button. If you do not open a port prior to installation, you cannot configure the MySQL server immediately after installation. Additionally, when running the MySQL Installation Wizard on Windows Vista, ensure that you are logged in as a user with administrative rights.

The process for starting the wizard depends on the contents of the installation package you download. If there is a [setup.exe](#) file present, double-click it to start the installation process. If there is an [.msi](#) file present, double-click it to start the installation process.

2.3.3.3. Choosing an Install Type

There are three installation types available: **Typical**, **Complete**, and **Custom**.

The **Typical** installation type installs the MySQL server, the [mysql](#) command-line client, and the command-line utilities. The command-line clients and utilities include [mysqldump](#), [myisamchk](#), and several other tools to help you manage the MySQL server.

The **Complete** installation type installs all components included in the installation package. The full installation package includes components such as the embedded server library, the benchmark suite, support scripts, and documentation.

The **Custom** installation type gives you complete control over which packages you wish to install and the installation path that is used. See [Section 2.3.3.4, “The Custom Install Dialog”](#), for more information on performing a custom install.

If you choose the **Typical** or **Complete** installation types and click the NEXT button, you advance to the confirmation screen to verify your choices and begin the installation. If you choose the **Custom** installation type and click the NEXT button, you advance to the custom installation dialog, described in [Section 2.3.3.4, “The Custom Install Dialog”](#).

2.3.3.4. The Custom Install Dialog

If you wish to change the installation path or the specific components that are installed by the MySQL Installation Wizard, choose the **Custom** installation type.

A tree view on the left side of the custom install dialog lists all available components. Components that are not installed have a red X icon; components that are installed have a gray icon. To change whether a component is installed, click on that component's icon and choose a new option from the drop-down list that appears.

You can change the default installation path by clicking the CHANGE... button to the right of the displayed installation path.

After choosing your installation components and installation path, click the NEXT button to advance to the confirmation dialog.

2.3.3.5. The Confirmation Dialog

Once you choose an installation type and optionally choose your installation components, you advance to the confirmation dialog. Your installation type and installation path are displayed for you to review.

To install MySQL if you are satisfied with your settings, click the INSTALL button. To change your settings, click the BACK button. To exit the MySQL Installation Wizard without installing MySQL, click the CANCEL button.

After installation is complete, you have the option of registering with the MySQL web site. Registration gives you access to post in the MySQL forums at [forums.mysql.com](#), along with the ability to report bugs at [bugs.mysql.com](#) and to subscribe to our newsletter. The final screen of the installer provides a summary of the installation and gives you the option to launch the MySQL Configuration Wizard, which you can use to create a configuration file, install the MySQL service, and configure security settings.

2.3.3.6. Changes Made by MySQL Installation Wizard

Once you click the INSTALL button, the MySQL Installation Wizard begins the installation process and makes certain changes to your system which are described in the sections that follow.

Changes to the Registry

The MySQL Installation Wizard creates one Windows registry key in a typical install situation, located in [HKEY_LOCAL_MACHINE\SOFTWARE\MySQL AB](#).

The MySQL Installation Wizard creates a key named after the major version of the server that is being installed, such as **MySQL Server 5.1**. It contains two string values, **Location** and **Version**. The **Location** string contains the path to the installation directory. In a default installation it contains **C:\Program Files\MySQL\MySQL Server 5.1**. The **Version** string contains the release number. For example, for an installation of MySQL Server 5.1.21-beta, the key contains a value of **5.1.21-beta**.

These registry keys are used to help external tools identify the installed location of the MySQL server, preventing a complete scan of the hard-disk to determine the installation path of the MySQL server. The registry keys are not required to run the server, and if you install MySQL using the **noinstall** Zip archive, the registry keys are not created.

Changes to the Start Menu

The MySQL Installation Wizard creates a new entry in the Windows **START** menu under a common MySQL menu heading named after the major version of MySQL that you have installed. For example, if you install MySQL 5.1, the MySQL Installation Wizard creates a MySQL Server 5.1 section in the **START** menu.

The following entries are created within the new **START** menu section:

- **MySQL Command Line Client:** This is a shortcut to the **mysql** command-line client and is configured to connect as the **root** user. The shortcut prompts for a **root** user password when you connect.
- **MySQL Server Instance Config Wizard:** This is a shortcut to the MySQL Configuration Wizard. Use this shortcut to configure a newly installed server, or to reconfigure an existing server.
- **MySQL Documentation:** This is a link to the MySQL server documentation that is stored locally in the MySQL server installation directory. This option is not available when the MySQL server is installed using the Essentials installation package.

Changes to the File System

The MySQL Installation Wizard by default installs the MySQL 5.1 server to **C:\Program Files\MySQL\MySQL Server 5.1**, where **Program Files** is the default location for applications in your system, and **5.1** is the major version of your MySQL server. This is the recommended location for the MySQL server, replacing the former default location **C:\mysql**.

By default, all MySQL applications are stored in a common directory at **C:\Program Files\MySQL**, where **Program Files** is the default location for applications in your Windows installation. A typical MySQL installation on a developer machine might look like this:

```
C:\Program Files\MySQL\MySQL Server 5.1
C:\Program Files\MySQL\MySQL Administrator 1.0
C:\Program Files\MySQL\MySQL Query Browser 1.0
```

This approach makes it easier to manage and maintain all MySQL applications installed on a particular system.

2.3.3.7. Upgrading MySQL with the Installation Wizard

The MySQL Installation Wizard can perform server upgrades automatically using the upgrade capabilities of MSI. That means you do not need to remove a previous installation manually before installing a new release. The installer automatically shuts down and removes the previous MySQL service before installing the new version.

Automatic upgrades are available only when upgrading between installations that have the same major and minor version numbers. For example, you can upgrade automatically from MySQL 4.1.5 to MySQL 4.1.6, but not from MySQL 5.0 to MySQL 5.1.

See [Section 2.3.14, “Upgrading MySQL on Windows”](#).

2.3.4. Using the Configuration Wizard

2.3.4.1. Introduction to the Configuration Wizard

The MySQL Configuration Wizard helps automate the process of configuring your server under Windows. The MySQL Configuration Wizard creates a custom **my.ini** file by asking you a series of questions and then applying your responses to a template to generate a **my.ini** file that is tuned to your installation.

The MySQL Configuration Wizard is included with the MySQL 5.1 server, and is currently available for Windows users only.

The MySQL Configuration Wizard is to a large extent the result of feedback that MySQL AB has received from many users over a period of several years. However, if you find that it lacks some feature important to you, please report it in our bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#).

2.3.4.2. Starting the MySQL Configuration Wizard

The MySQL Configuration Wizard is typically launched from the MySQL Installation Wizard, as the MySQL Installation Wizard exits. You can also launch the MySQL Configuration Wizard by clicking the MySQL Server Instance Config Wizard entry in the MySQL section of the Windows [START](#) menu.

Alternatively, you can navigate to the `bin` directory of your MySQL installation and launch the `MySQLInstanceConfig.exe` file directly.

Note

If you chose not to open a port prior to installing MySQL on Windows Vista, you can choose to use the MySQL Server Configuration Wizard after installation. However, you must open a port in the Windows Firewall. To do this see the instructions given in [Section 2.3.3.2, “Downloading and Starting the MySQL Installation Wizard”](#). Rather than opening a port, you also have the option of adding MySQL as a program that bypasses the Windows Firewall. One or the other option is sufficient — you need not do both. Additionally, when running the MySQL Server Configuration Wizard on Windows Vista ensure that you are logged in as a user with administrative rights.

2.3.4.3. Choosing a Maintenance Option

If the MySQL Configuration Wizard detects an existing `my.ini` file, you have the option of either reconfiguring your existing server, or removing the server instance by deleting the `my.ini` file and stopping and removing the MySQL service.

To reconfigure an existing server, choose the Re-configure Instance option and click the NEXT button. Your existing `my.ini` file is renamed to `mytimestamp.ini.bak`, where `timestamp` is the date and time at which the existing `my.ini` file was created. To remove the existing server instance, choose the Remove Instance option and click the NEXT button.

If you choose the Remove Instance option, you advance to a confirmation window. Click the EXECUTE button. The MySQL Configuration Wizard stops and removes the MySQL service, and then deletes the `my.ini` file. The server installation and its `data` folder are not removed.

If you choose the Re-configure Instance option, you advance to the [CONFIGURATION TYPE](#) dialog where you can choose the type of installation that you wish to configure.

2.3.4.4. Choosing a Configuration Type

When you start the MySQL Configuration Wizard for a new MySQL installation, or choose the Re-configure Instance option for an existing installation, you advance to the [CONFIGURATION TYPE](#) dialog.

There are two configuration types available: Detailed Configuration and Standard Configuration. The Standard Configuration option is intended for new users who want to get started with MySQL quickly without having to make many decisions about server configuration. The Detailed Configuration option is intended for advanced users who want more fine-grained control over server configuration.

If you are new to MySQL and need a server configured as a single-user developer machine, the Standard Configuration should suit your needs. Choosing the Standard Configuration option causes the MySQL Configuration Wizard to set all configuration options automatically with the exception of Service Options and Security Options.

The Standard Configuration sets options that may be incompatible with systems where there are existing MySQL installations. If you have an existing MySQL installation on your system in addition to the installation you wish to configure, the Detailed Configuration option is recommended.

To complete the Standard Configuration, please refer to the sections on Service Options and Security Options in [Section 2.3.4.11, “The Service Options Dialog”](#), and [Section 2.3.4.12, “The Security Options Dialog”](#), respectively.

2.3.4.5. The Server Type Dialog

There are three different server types available to choose from. The server type that you choose affects the decisions that the MySQL Configuration Wizard makes with regard to memory, disk, and processor usage.

- **Developer Machine:** Choose this option for a typical desktop workstation where MySQL is intended only for personal use. It is assumed that many other desktop applications are running. The MySQL server is configured to use minimal system resources.
- **Server Machine:** Choose this option for a server machine where the MySQL server is running alongside other server applications such as FTP, email, and Web servers. The MySQL server is configured to use a moderate portion of the system resources.
- **Dedicated MySQL Server Machine:** Choose this option for a server machine that is intended to run only the MySQL server. It is assumed that no other applications are running. The MySQL server is configured to use all available system resources.

Note

By selecting one of the preconfigured configurations, the values and settings of various options in your `my.cnf` or `my.ini` will be altered accordingly. The default values and options as described in the reference manual may therefore be different to the options and values that were created during the execution of the configuration wizard.

2.3.4.6. The Database Usage Dialog

The **DATABASE USAGE** dialog allows you to indicate the storage engines that you expect to use when creating MySQL tables. The option you choose determines whether the **InnoDB** storage engine is available and what percentage of the server resources are available to **InnoDB**.

- **Multifunctional Database:** This option enables both the **InnoDB** and **MyISAM** storage engines and divides resources evenly between the two. This option is recommended for users who use both storage engines on a regular basis.
- **Transactional Database Only:** This option enables both the **InnoDB** and **MyISAM** storage engines, but dedicates most server resources to the **InnoDB** storage engine. This option is recommended for users who use **InnoDB** almost exclusively and make only minimal use of **MyISAM**.
- **Non-Transactional Database Only:** This option disables the **InnoDB** storage engine completely and dedicates all server resources to the **MyISAM** storage engine. This option is recommended for users who do not use **InnoDB**.

2.3.4.7. The InnoDB Tablespace Dialog

Some users may want to locate the **InnoDB** tablespace files in a different location than the MySQL server data directory. Placing the tablespace files in a separate location can be desirable if your system has a higher capacity or higher performance storage device available, such as a RAID storage system.

To change the default location for the **InnoDB** tablespace files, choose a new drive from the drop-down list of drive letters and choose a new path from the drop-down list of paths. To create a custom path, click the ... button.

If you are modifying the configuration of an existing server, you must click the **MODIFY** button before you change the path. In this situation you must move the existing tablespace files to the new location manually before starting the server.

2.3.4.8. The Concurrent Connections Dialog

To prevent the server from running out of resources, it is important to limit the number of concurrent connections to the MySQL server that can be established. The **CONCURRENT CONNECTIONS** dialog allows you to choose the expected usage of your server, and sets the limit for concurrent connections accordingly. It is also possible to set the concurrent connection limit manually.

- **Decision Support (DSS)/OLAP:** Choose this option if your server does not require a large number of concurrent connections. The maximum number of connections is set at 100, with an average of 20 concurrent connections assumed.
- **Online Transaction Processing (OLTP):** Choose this option if your server requires a large number of concurrent connections. The maximum number of connections is set at 500.
- **Manual Setting:** Choose this option to set the maximum number of concurrent connections to the server manually. Choose the number of concurrent connections from the drop-down box provided, or enter the maximum number of connections into the drop-down box if the number you desire is not listed.

2.3.4.9. The Networking and Strict Mode Options Dialog

Use the [NETWORKING OPTIONS](#) dialog to enable or disable TCP/IP networking and to configure the port number that is used to connect to the MySQL server.

TCP/IP networking is enabled by default. To disable TCP/IP networking, uncheck the box next to the Enable TCP/IP Networking option.

Port 3306 is used by default. To change the port used to access MySQL, choose a new port number from the drop-down box or type a new port number directly into the drop-down box. If the port number you choose is in use, you are prompted to confirm your choice of port number.

Set the [SERVER SQL MODE](#) to either enable or disable strict mode. Enabling strict mode (default) makes MySQL behave more like other database management systems. *If you run applications that rely on MySQL's old "forgiving" behavior, make sure to either adapt those applications or to disable strict mode.* For more information about strict mode, see [Section 5.2.6, "SQL Modes"](#).

2.3.4.10. The Character Set Dialog

The MySQL server supports multiple character sets and it is possible to set a default server character set that is applied to all tables, columns, and databases unless overridden. Use the [CHARACTER SET](#) dialog to change the default character set of the MySQL server.

- **Standard Character Set:** Choose this option if you want to use [latin1](#) as the default server character set. [latin1](#) is used for English and many Western European languages.
- **Best Support For Multilingualism:** Choose this option if you want to use [utf8](#) as the default server character set. This is a Unicode character set that can store characters from many different languages.
- **Manual Selected Default Character Set / Collation:** Choose this option if you want to pick the server's default character set manually. Choose the desired character set from the provided drop-down list.

2.3.4.11. The Service Options Dialog

On Windows platforms, the MySQL server can be installed as a Windows service. When installed this way, the MySQL server can be started automatically during system startup, and even restarted automatically by Windows in the event of a service failure.

The MySQL Configuration Wizard installs the MySQL server as a service by default, using the service name [MySQL](#). If you do not wish to install the service, uncheck the box next to the Install As Windows Service option. You can change the service name by picking a new service name from the drop-down box provided or by entering a new service name into the drop-down box.

To install the MySQL server as a service but not have it started automatically at startup, uncheck the box next to the Launch the MySQL Server Automatically option.

2.3.4.12. The Security Options Dialog

It is strongly recommended that you set a [root](#) password for your MySQL server, and the MySQL Configuration Wizard requires by default that you do so. If you do not wish to set a [root](#) password, uncheck the box next to the Modify Security Settings option.

To set the [root](#) password, enter the desired password into both the New root password and Confirm boxes. If you are reconfiguring an existing server, you need to enter the existing [root](#) password into the Current root password box.

To prevent [root](#) logins from across the network, check the box next to the Root may only connect from localhost option. This increases the security of your [root](#) account.

To create an anonymous user account, check the box next to the Create An Anonymous Account option. Creating an anonymous account can decrease server security and cause login and permission difficulties. For this reason, it is not recommended.

2.3.4.13. The Confirmation Dialog

The final dialog in the MySQL Configuration Wizard is the [CONFIRMATION DIALOG](#). To start the configuration process, click the EXECUTE button. To return to a previous dialog, click the BACK button. To exit the MySQL Configuration Wizard without configuring the server, click the CANCEL button.

After you click the EXECUTE button, the MySQL Configuration Wizard performs a series of tasks and displays the progress onscreen as the tasks are performed.

The MySQL Configuration Wizard first determines configuration file options based on your choices using a template prepared by MySQL AB developers and engineers. This template is named `my-template.ini` and is located in your server installation directory.

The MySQL Configuration Wizard then writes these options to a `my.ini` file. The final location of the `my.ini` file is displayed next to the **WRITE CONFIGURATION FILE** task.

If you chose to create a service for the MySQL server, the MySQL Configuration Wizard creates and starts the service. If you are reconfiguring an existing service, the MySQL Configuration Wizard restarts the service to apply your configuration changes.

If you chose to set a `root` password, the MySQL Configuration Wizard connects to the server, sets your new `root` password and applies any other security settings you may have selected.

After the MySQL Configuration Wizard has completed its tasks, it displays a summary. Click the **FINISH** button to exit the MySQL Configuration Wizard.

2.3.4.14. The Location of the my.ini File

The MySQL Configuration Wizard places the `my.ini` file in the installation directory for the MySQL server. This helps associate configuration files with particular server instances.

To ensure that the MySQL server knows where to look for the `my.ini` file, an argument similar to this is passed to the MySQL server as part of the service installation:

```
--defaults-file="C:\Program Files\MySQL\MySQL Server 5.1\my.ini"
```

Here, `C:\Program Files\MySQL\MySQL Server 5.1` is replaced with the installation path to the MySQL Server. The `--defaults-file` option instructs the MySQL server to read the specified file for configuration options when it starts.

2.3.4.15. Editing the my.ini File

To modify the `my.ini` file, open it with a text editor and make any necessary changes. You can also modify the server configuration with the [MySQL Administrator](#) utility.

MySQL clients and utilities such as the `mysql` and `mysqldump` command-line clients are not able to locate the `my.ini` file located in the server installation directory. To configure the client and utility applications, create a new `my.ini` file in the `C:\WINDOWS` directory (whichever is applicable to your Windows version).

2.3.5. Installing MySQL from a Noinstall Zip Archive

Users who are installing from the Noinstall package can use the instructions in this section to manually install MySQL. The process for installing MySQL from a Zip archive is as follows:

1. Extract the archive to the desired install directory
2. Create an option file
3. Choose a MySQL server type
4. Start the MySQL server
5. Secure the default user accounts

This process is described in the sections that follow.

2.3.6. Extracting the Install Archive

To install MySQL manually, do the following:

1. If you are upgrading from a previous version please refer to [Section 2.3.14, “Upgrading MySQL on Windows”](#), before beginning the upgrade process.

2. Make sure that you are logged in as a user with administrator privileges.
3. Choose an installation location. Traditionally, the MySQL server is installed in `C:\mysql`. The MySQL Installation Wizard installs MySQL under `C:\Program Files\MySQL`. If you do not install MySQL at `C:\mysql`, you must specify the path to the install directory during startup or in an option file. See [Section 2.3.7, “Creating an Option File”](#).
4. Extract the install archive to the chosen installation location using your preferred Zip archive tool. Some tools may extract the archive to a folder within your chosen installation location. If this occurs, you can move the contents of the subfolder into the chosen installation location.

2.3.7. Creating an Option File

If you need to specify startup options when you run the server, you can indicate them on the command line or place them in an option file. For options that are used every time the server starts, you may find it most convenient to use an option file to specify your MySQL configuration. This is particularly true under the following circumstances:

- The installation or data directory locations are different from the default locations (`C:\Program Files\MySQL\MySQL Server 5.1` and `C:\Program Files\MySQL\MySQL Server 5.1\data`).
- You need to tune the server settings.

When the MySQL server starts on Windows, it looks for options in two files: the `my.ini` file in the Windows directory, and the `C:\my.cnf` file. The Windows directory typically is named something like `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
C:\> echo %WINDIR%
```

MySQL looks for options first in the `my.ini` file, and then in the `my.cnf` file. However, to avoid confusion, it's best if you use only one file. If your PC uses a boot loader where `C:` is not the boot drive, your only option is to use the `my.ini` file. Whichever option file you use, it must be a plain text file.

You can also make use of the example option files included with your MySQL distribution; see [Section 4.3.2.1, “Preconfigured Option Files”](#).

An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in `E:\mysql` and the data directory is in `E:\mydata\data`, you can create an option file containing a `[mysqld]` section to specify values for the `basedir` and `datadir` parameters:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=E:/mydata/data
```

Note that Windows pathnames are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, you must double them:

```
[mysqld]
# set basedir to your installation path
basedir=E:\\mysql
# set datadir to the location of your data directory
datadir=E:\\mydata\\data
```

MySQL Enterprise

For expert advice on the start-up options appropriate to your circumstances, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

On Windows, the MySQL installer places the data directory directly under the directory where you install MySQL. If you would like to use a data directory in a different location, you should copy the entire contents of the `data` directory to the new location. For example, if MySQL is installed in `C:\Program Files\MySQL\MySQL Server 5.1`, the data directory is by default in `C:\Program Files\MySQL\MySQL Server 5.1\data`. If you want to use `E:\mydata` as the data directory instead, you must do two things:

1. Move the entire `data` directory and all of its contents from `C:\Program Files\MySQL\MySQL Server 5.1\data` to `E:\mydata`.
2. Use a `--datadir` option to specify the new data directory location each time you start the server.

2.3.8. Selecting a MySQL Server Type

The following table shows the available servers for Windows in MySQL 5.1.

Binary	Description
<code>mysqld-nt</code>	Optimized binary with named-pipe support
<code>mysqld</code>	Optimized binary without named-pipe support
<code>mysqld-debug</code>	Like <code>mysqld-nt</code> , but compiled with full debugging and automatic memory allocation checking

All of the preceding binaries are optimized for modern Intel processors, but should work on any Intel i386-class or higher processor.

Each of the servers in a distribution support the same set of storage engines. The `SHOW ENGINES` statement displays which engines a given server supports.

All Windows MySQL 5.1 servers have support for symbolic linking of database directories.

MySQL supports TCP/IP on all Windows platforms. MySQL servers on Windows support named pipes as indicated in the following list. However, the default is to use TCP/IP regardless of platform. (Named pipes are slower than TCP/IP in many Windows configurations.)

Use of named pipes is subject to these conditions:

- Named pipes are enabled only if you start the server with the `--enable-named-pipe` option. It is necessary to use this option explicitly because some users have experienced problems with shutting down the MySQL server when named pipes were used.
- Named-pipe connections are allowed only by the `mysqld-nt` and `mysqld-debug` servers.

Note

Most of the examples in this manual use `mysqld` as the server name. If you choose to use a different server, such as `mysqld-nt`, make the appropriate substitutions in the commands that are shown in the examples.

2.3.9. Starting the Server for the First Time

This section gives a general overview of starting the MySQL server. The following sections provide more specific information for starting the MySQL server from the command line or as a Windows service.

The information here applies primarily if you installed MySQL using the `Noinstall` version, or if you wish to configure and test MySQL manually rather than with the GUI tools.

The examples in these sections assume that MySQL is installed under the default location of `C:\Program Files\MySQL\MySQL Server 5.1`. Adjust the pathnames shown in the examples if you have MySQL installed in a different location.

Clients have two options. They can use TCP/IP, or they can use a named pipe if the server supports named-pipe connections.

MySQL for Windows also supports shared-memory connections if the server is started with the `--shared-memory` option. Clients can connect through shared memory by using the `--protocol=memory` option.

For information about which server binary to run, see [Section 2.3.8, “Selecting a MySQL Server Type”](#).

Testing is best done from a command prompt in a console window (or “DOS window”). In this way you can have the server display status messages in the window where they are easy to see. If something is wrong with your configuration, these messages make it easier for you to identify and fix any problems.

To start the server, enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld" --console
```

For a server that includes **InnoDB** support, you should see the messages similar to those following as it starts (the pathnames and sizes may differ):

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25 InnoDB: Started
```

When the server finishes its startup sequence, you should see something like this, which indicates that the server is ready to service client connections:

```
mysqld: ready for connections
Version: '5.1.21-beta' socket: '' port: 3306
```

The server continues to write to the console any further diagnostic output it produces. You can open a new console window in which to run client programs.

If you omit the `--console` option, the server writes diagnostic output to the error log in the data directory (`C:\Program Files\MySQL\MySQL Server 5.1\data` by default). The error log is the file with the `.err` extension.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

2.3.10. Starting MySQL from the Windows Command Line

The MySQL server can be started manually from the command line. This can be done on any version of Windows.

To start the `mysqld` server from the command line, you should start a console window (or “DOS window”) and enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld"
```

The path to `mysqld` may vary depending on the install location of MySQL on your system.

You can stop the MySQL server by executing this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin" -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

This command invokes the MySQL administrative utility `mysqladmin` to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

If `mysqld` doesn't start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the `C:\Program Files\MySQL\MySQL Server 5.1\data` directory. It is the file with a suffix of `.err`. You can also try to start the server as `mysqld --console`; in this case, you may get some useful information on the screen that may help solve the problem.

The last option is to start `mysqld` with the `--standalone` and `--debug` options. In this case, `mysqld` writes a log file `C:\mysqld.trace` that should contain the reason why `mysqld` doesn't start. See [MySQL Internals: Porting](#).

Use `mysqld --verbose --help` to display all the options that `mysqld` understands.

2.3.11. Starting MySQL as a Windows Service

On Windows, the recommended way to run MySQL is to install it as a Windows service, whereby MySQL starts and stops automatically when Windows starts and stops. A MySQL server installed as a service can also be controlled from the command line using [NET](#) commands, or with the graphical [Services](#) utility. Generally, to install MySQL as a Windows service you should be logged in using an account that has administrator rights.

The [Services](#) utility (the Windows [Service Control Manager](#)) can be found in the Windows Control Panel (under Administrative Tools on Windows 2000, XP, Vista and Server 2003). To avoid conflicts, it is advisable to close the [Services](#) utility while performing server installation or removal operations from the command line.

Before installing MySQL as a Windows service, you should first stop the current server if it is running by using the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin" -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

This command invokes the MySQL administrative utility `mysqladmin` to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

Install the server as a service using this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld" --install
```

The service-installation command does not start the server. Instructions for that are given later in this section.

To make it easier to invoke MySQL programs, you can add the pathname of the MySQL `bin` directory to your Windows system [PATH](#) environment variable:

- On the Windows desktop, right-click on the My Computer icon, and select Properties
- Next select the Advanced tab from the [SYSTEM PROPERTIES](#) menu that appears, and click the ENVIRONMENT VARIABLES button.
- Under **SYSTEM VARIABLES**, select Path, and then click the EDIT button. The [EDIT SYSTEM VARIABLE](#) dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **VARIABLE VALUE**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete pathname of your MySQL `bin` directory (for example, `C:\Program Files\MySQL\MySQL Server 5.1\bin`). Note that there should be a semicolon separating this path from any values present in this field. Dismiss this dialogue, and each dialogue in turn, by clicking OK until all of the dialogues that were opened have been dismissed. You should now be able to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the `mysql` client, and all MySQL command-line utilities such as `mysqladmin` and `mysqldump`.

You should not add the MySQL `bin` directory to your Windows [PATH](#) if you are running multiple MySQL servers on the same machine.

Warning

You must exercise great care when editing your system [PATH](#) by hand; accidental deletion or modification of any portion of the existing [PATH](#) value can leave you with a malfunctioning or even unusable system.

The following additional arguments can be used in MySQL 5.1 when installing the service:

- You can specify a service name immediately following the `--install` option. The default service name is `MySQL`.
- If a service name is given, it can be followed by a single option. By convention, this should be `--defaults-file=file_name` to specify the name of an option file from which the server should read options when it starts.

It is possible to use a single option other than `--defaults-file`, but this is discouraged. `--defaults-file` is more flexible because it enables you to specify multiple startup options for the server by placing them in the named option file.
- You can also specify a `--local-service` option following the service name. This causes the server to run using the `LocalService` Windows account that has limited system privileges. This account is available only for Windows XP or newer. If both `--defaults-file` and `--local-service` are given following the service name, they can be in any order.

For a MySQL server that is installed as a Windows service, the following rules determine the service name and option files that the server uses:

- If the service-installation command specifies no service name or the default service name (`MySQL`) following the `--install` option, the server uses the a service name of `MySQL` and reads options from the `[mysqld]` group in the standard option files.
- If the service-installation command specifies a service name other than `MySQL` following the `--install` option, the server uses that service name. It reads options from the group that has the same name as the service, and reads options from the standard option files.

The server also reads options from the `[mysqld]` group from the standard option files. This allows you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group with the same name as a service for use by the server installed with that service name.

- If the service-installation command specifies a `--defaults-file` option after the service name, the server reads options only from the `[mysqld]` group of the named file and ignores the standard option files.

As a more complex example, consider the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld"
      --install MySQL --defaults-file=C:\my-opts.cnf
```

Here, the default service name (`MySQL`) is given after the `--install` option. If no `--defaults-file` option had been given, this command would have the effect of causing the server to read the `[mysqld]` group from the standard option files. However, because the `--defaults-file` option is present, the server reads options from the `[mysqld]` option group, and only from the named file.

You can also specify options as Start parameters in the Windows `Services` utility before you start the MySQL service.

Once a MySQL server has been installed as a service, Windows starts the service automatically whenever Windows starts. The service also can be started immediately from the `Services` utility, or by using a `NET START MySQL` command. The `NET` command is not case sensitive.

When run as a service, `mysqld` has no access to a console window, so no messages can be seen there. If `mysqld` does not start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the MySQL data directory (for example, `C:\Program Files\MySQL\MySQL Server 5.1\data`). It is the file with a suffix of `.err`.

When a MySQL server has been installed as a service, and the service is running, Windows stops the service automatically when Windows shuts down. The server also can be stopped manually by using the `Services` utility, the `NET STOP MySQL` command, or the `mysqladmin shutdown` command.

You also have the choice of installing the server as a manual service if you do not wish for the service to be started automatically during the boot process. To do this, use the `--install-manual` option rather than the `--install` option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld" --install-manual
```

To remove a server that is installed as a service, first stop it if it is running by executing `NET STOP MySQL`. Then use the `--remove` option to remove it:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld" --remove
```

If `mysqld` is not running as a service, you can start it from the command line. For instructions, see [Section 2.3.10, “Starting MySQL from the Windows Command Line”](#).

Please see [Section 2.3.13, “Troubleshooting a MySQL Installation Under Windows”](#), if you encounter difficulties during installation.

2.3.12. Testing The MySQL Installation

You can test whether the MySQL server is working by executing any of the following commands:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqlshow"  
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqlshow" -u root mysql  
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin" version status proc  
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql" test
```

If `mysqld` is slow to respond to TCP/IP connections from client programs, there is probably a problem with your DNS. In this case, start `mysqld` with the `--skip-name-resolve` option and use only `localhost` and IP numbers in the `Host` column of the MySQL grant tables.

You can force a MySQL client to use a named-pipe connection rather than TCP/IP by specifying the `--pipe` or `--protocol=PIPE` option, or by specifying `.` (period) as the host name. Use the `--socket` option to specify the name of the pipe if you do not want to use the default pipe name.

Note that if you have set a password for the `root` account, deleted the anonymous account, or created a new user account, then you must use the appropriate `-u` and `-p` options with the commands shown above in order to connect with the MySQL Server. See [Section 5.7.4, “Connecting to the MySQL Server”](#).

For more information about `mysqlshow`, see [Section 7.16, “mysqlshow — Display Database, Table, and Column Information”](#).

2.3.13. Troubleshooting a MySQL Installation Under Windows

When installing and running MySQL for the first time, you may encounter certain errors that prevent the MySQL server from starting. The purpose of this section is to help you diagnose and correct some of these errors.

Your first resource when troubleshooting server issues is the error log. The MySQL server uses the error log to record information relevant to the error that prevents the server from starting. The error log is located in the data directory specified in your `my.ini` file. The default data directory location is `C:\Program Files\MySQL\MySQL Server 5.1\data`. See [Section 5.11.2, “The Error Log”](#).

Another source of information regarding possible errors is the console messages displayed when the MySQL service is starting. Use the `NET START MySQL` command from the command line after installing `mysqld` as a service to see any error messages regarding the starting of the MySQL server as a service. See [Section 2.3.11, “Starting MySQL as a Windows Service”](#).

The following examples show other common error messages you may encounter when installing MySQL and starting the server for the first time:

- If the MySQL server cannot find the `mysql` privileges database or other critical files, you may see these messages:

```
System error 1067 has occurred.  
Fatal error: Can't open privilege tables: Table 'mysql.host' doesn't exist
```

These messages often occur when the MySQL base or data directories are installed in different locations than the default locations (`C:\Program Files\MySQL\MySQL Server 5.1` and `C:\Program Files\MySQL\MySQL Server 5.1\data`, respectively).

This situation may occur when MySQL is upgraded and installed to a new location, but the configuration file is not updated to reflect the new location. In addition, there may be old and new configuration files that conflict. Be sure to delete or rename any old configuration files when upgrading MySQL.

If you have installed MySQL to a directory other than `C:\Program Files\MySQL\MySQL Server 5.1`, you need to ensure that the MySQL server is aware of this through the use of a configuration (`my.ini`) file. The `my.ini` file needs to be located in your Windows directory, typically `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable by issuing the following command from the command prompt:

```
C:\> echo %WINDIR%
```

An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in `E:\mysql` and the data directory is `D:\MySQLdata`, you can create the option file and set up a `[mysqld]` section to specify values for the `basedir` and `datadir` parameters:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=D:/MySQLdata
```

Note that Windows pathnames are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, you must double them:

```
[mysqld]
# set basedir to your installation path
basedir=C:\\Program Files\\MySQL\\MySQL Server 5.1
# set datadir to the location of your data directory
datadir=D:\\MySQLdata
```

If you change the `datadir` value in your MySQL configuration file, you must move the contents of the existing MySQL data directory before restarting the MySQL server.

See [Section 2.3.7, “Creating an Option File”](#).

- If you reinstall or upgrade MySQL without first stopping and removing the existing MySQL service and install MySQL using the MySQL Configuration Wizard, you may see this error:

```
Error: Cannot create Windows service for MySQL. Error: 0
```

This occurs when the Configuration Wizard tries to install the service and finds an existing service with the same name.

One solution to this problem is to choose a service name other than `mysql` when using the configuration wizard. This allows the new service to be installed correctly, but leaves the outdated service in place. Although this is harmless, it is best to remove old services that are no longer in use.

To permanently remove the old `mysql` service, execute the following command as a user with administrative privileges, on the command-line:

```
C:\> sc delete mysql
[SC] DeleteService SUCCESS
```

If the `sc` utility is not available for your version of Windows, download the `delsrv` utility from <http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp> and use the `delsrv mysql` syntax.

2.3.14. Upgrading MySQL on Windows

This section lists some of the steps you should take when upgrading MySQL on Windows.

1. Review [Section 2.11, “Upgrading MySQL”](#), for additional information on upgrading MySQL that is not specific to Windows.
2. You should always back up your current MySQL installation before performing an upgrade. See [Section 5.9.1, “Database Backups”](#).
3. Download the latest Windows distribution of MySQL from <http://dev.mysql.com/downloads/>.
4. Before upgrading MySQL, you must stop the server. If the server is installed as a service, stop the service with the following command from the command prompt:

```
C:\> NET STOP MySQL
```

If you are not running the MySQL server as a service, use the following command to stop it:


```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin" -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

- When upgrading to MySQL 5.1 from a version previous to 4.1.5, or when upgrading from a version of MySQL installed from a Zip archive to a version of MySQL installed with the MySQL Installation Wizard, you must manually remove the previous installation and MySQL service (if the server is installed as a service).

To remove the MySQL service, use the following command:

```
C:\> C:\mysql\bin\mysqld --remove
```

If you do not remove the existing service, the MySQL Installation Wizard may fail to properly install the new MySQL service.

- If you are using the MySQL Installation Wizard, start the wizard as described in [Section 2.3.3, “Using the MySQL Installation Wizard”](#).
- If you are installing MySQL from a Zip archive, extract the archive. You may either overwrite your existing MySQL installation (usually located at `C:\mysql`), or install it into a different directory, such as `C:\mysql5`. Overwriting the existing installation is recommended.
- If you were running MySQL as a Windows service and you had to remove the service earlier in this procedure, reinstall the service. (See [Section 2.3.11, “Starting MySQL as a Windows Service”](#).)
- Restart the server. For example, use `NET START MySQL` if you run MySQL as a service, or invoke `mysqld` directly otherwise.
- If you encounter errors, see [Section 2.3.13, “Troubleshooting a MySQL Installation Under Windows”](#).

2.3.15. MySQL on Windows Compared to MySQL on Unix

MySQL for Windows has proven itself to be very stable. The Windows version of MySQL has the same features as the corresponding Unix version, with the following exceptions:

- **Limited number of ports**

Windows systems have about 4,000 ports available for client connections, and after a connection on a port closes, it takes two to four minutes before the port can be reused. In situations where clients connect to and disconnect from the server at a high rate, it is possible for all available ports to be used up before closed ports become available again. If this happens, the MySQL server appears to be unresponsive even though it is running. Note that ports may be used by other applications running on the machine as well, in which case the number of ports available to MySQL is lower.

For more information about this problem, see <http://support.microsoft.com/default.aspx?scid=kb;en-us;196271>.

- **Concurrent reads**

MySQL depends on the `pread()` and `pwrite()` system calls to be able to mix `INSERT` and `SELECT`. Currently, we use mutexes to emulate `pread()` and `pwrite()`. We intend to replace the file level interface with a virtual interface in the future so that we can use the `readfile()/writefile()` interface to get more speed. The current implementation limits the number of open files that MySQL 5.1 can use to 2,048, which means that you cannot run as many concurrent threads on Windows as on Unix.

- **Blocking read**

MySQL uses a blocking read for each connection. That has the following implications if named-pipe connections are enabled:

- A connection is not disconnected automatically after eight hours, as happens with the Unix version of MySQL.
- If a connection hangs, it is not possible to break it without killing MySQL.

- `mysqladmin kill` does not work on a sleeping connection.
- `mysqladmin shutdown` cannot abort as long as there are sleeping connections.

We plan to fix this problem in the future.

- **ALTER TABLE**

While you are executing an `ALTER TABLE` statement, the table is locked from being used by other threads. This has to do with the fact that on Windows, you can't delete a file that is in use by another thread. In the future, we may find some way to work around this problem.

- **DROP TABLE**

`DROP TABLE` on a table that is in use by a `MERGE` table does not work on Windows because the `MERGE` handler does the table mapping hidden from the upper layer of MySQL. Because Windows does not allow dropping files that are open, you first must flush all `MERGE` tables (with `FLUSH TABLES`) or drop the `MERGE` table before dropping the table.

- **DATA DIRECTORY and INDEX DIRECTORY**

The `DATA DIRECTORY` and `INDEX DIRECTORY` options for `CREATE TABLE` are ignored on Windows, because Windows doesn't support symbolic links. These options also are ignored on systems that have a non-functional `realpath()` call.

- **DROP DATABASE**

You cannot drop a database that is in use by some thread.

- **Case-insensitive names**

Filenames are not case sensitive on Windows, so MySQL database and table names are also not case sensitive on Windows. The only restriction is that database and table names must be specified using the same case throughout a given statement. See [Section 8.2.2, “Identifier Case Sensitivity”](#).

- **The ‘\’ pathname separator character**

Pathname components in Windows are separated by the ‘\’ character, which is also the escape character in MySQL. If you are using `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, use Unix-style filenames with ‘/’ characters:

```
mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
```

Alternatively, you must double the ‘\’ character:

```
mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
```

- **Problems with pipes**

Pipes do not work reliably from the Windows command-line prompt. If the pipe includes the character `^Z / CHAR(24)`, Windows thinks that it has encountered end-of-file and aborts the program.

This is mainly a problem when you try to apply a binary log as follows:

```
C:\> mysqlbinlog binary_log_file | mysql --user=root
```

If you have a problem applying the log and suspect that it is because of a `^Z / CHAR(24)` character, you can use the following workaround:

```
C:\> mysqlbinlog binary_log_file --result-file=/tmp/bin.sql
C:\> mysql --user=root --execute "source /tmp/bin.sql"
```

The latter command also can be used to reliably read in any SQL file that may contain binary data.

- **Access denied for user error**

If MySQL cannot resolve your hostname properly, you may get the following error when you attempt to run a MySQL client program to connect to a server running on the same machine:

```
Access denied for user 'some_user'@'unknown'
to database 'mysql'
```

To fix this problem, you should create a file named `\windows\hosts` containing the following information:

```
127.0.0.1      localhost
```

Here are some open issues for anyone who might want to help us improve MySQL on Windows:

- Add macros to use the faster thread-safe increment/decrement methods provided by Windows.

2.4. Installing MySQL from RPM Packages on Linux

The recommended way to install MySQL on RPM-based Linux distributions is by using the RPM packages. The RPMs provided by MySQL AB to the community should work on all versions of Linux that support RPM packages and use `glibc` 2.3. To obtain RPM packages, see [Section 2.1.3, “How to Get MySQL”](#).

For non-RPM Linux distributions, you can install MySQL using a `.tar.gz` package. See [Section 2.8, “Installing MySQL from tar.gz Packages on Other Unix-Like Systems”](#).

MySQL AB does provide some platform-specific RPMs; the difference between a platform-specific RPM and a generic RPM is that a platform-specific RPM is built on the targeted platform and is linked dynamically whereas a generic RPM is linked statically with `LinuxThreads`.

Note

RPM distributions of MySQL often are provided by other vendors. Be aware that they may differ in features and capabilities from those built by MySQL AB, and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead.

If you have problems with an RPM file (for example, if you receive the error `Sorry, the host 'xxxx' could not be looked up`), see [Section 2.13.1.2, “Linux Binary Distribution Notes”](#).

In most cases, you need to install only the `MySQL-server` and `MySQL-client` packages to get a functional MySQL installation. The other packages are not required for a standard installation.

If you get a dependency failure when trying to install MySQL packages (for example, `error: removing these packages would break dependencies: libmysqlclient.so.10 is needed by ...`), you should also install the `MySQL-shared-compat` package, which includes both the shared libraries for backward compatibility (`libmysqlclient.so.12` for MySQL 4.0 and `libmysqlclient.so.10` for MySQL 3.23).

Some Linux distributions still ship with MySQL 3.23 and they usually link applications dynamically to save disk space. If these shared libraries are in a separate package (for example, `MySQL-shared`), it is sufficient to simply leave this package installed and just upgrade the MySQL server and client packages (which are statically linked and do not depend on the shared libraries). For distributions that include the shared libraries in the same package as the MySQL server (for example, Red Hat Linux), you could either install our 3.23 `MySQL-shared` RPM, or use the `MySQL-shared-compat` package instead. (Do not install both.)

The RPM packages shown in the following list are available. The names shown here use a suffix of `.glibc23.i386.rpm`, but particular packages can have different suffixes, as described later.

- `MySQL-server-VERSION.glibc23.i386.rpm`

The MySQL server. You need this unless you only want to connect to a MySQL server running on another machine.

- `MySQL-client-VERSION.glibc23.i386.rpm`

The standard MySQL client programs. You probably always want to install this package.

- [MySQL-devel-VERSION.glibc23.i386.rpm](#)

The libraries and include files that are needed if you want to compile other MySQL clients, such as the Perl modules.

- [MySQL-debuginfo-VERSION.glibc23.i386.rpm](#)

This package contains debugging information. [debuginfo](#) RPMs are never needed to use MySQL software; this is true both for the server and for client programs. However, they contain additional information that might be needed by a debugger to analyze a crash.

- [MySQL-shared-VERSION.glibc23.i386.rpm](#)

This package contains the shared libraries ([libmysqlclient.so*](#)) that certain languages and applications need to dynamically load and use MySQL. It contains single-threaded and thread-safe libraries. If you install this package, do not install the [MySQL-shared-compat](#) package.

- [MySQL-shared-compat-VERSION.glibc23.i386.rpm](#)

This package includes the shared libraries for MySQL 3.23, 4.0, 4.1, and 5.1. It contains single-threaded and thread-safe libraries. Install this package instead of [MySQL-shared](#) if you have applications installed that are dynamically linked against older versions of MySQL but you want to upgrade to the current version without breaking the library dependencies.

- [MySQL-embedded-VERSION.glibc23.i386.rpm](#)

The embedded MySQL server library.

- [MySQL-ndb-management-VERSION.glibc23.i386.rpm](#), [MySQL-ndb-storage-VERSION.glibc23.i386.rpm](#), [MySQL-ndb-tools-VERSION.glibc23.i386.rpm](#), [MySQL-ndb-extra-VERSION.glibc23.i386.rpm](#)

Packages that contain additional files for MySQL Cluster installations.

- [MySQL-test-VERSION.glibc23.i386.rpm](#)

This package includes the MySQL test suite.

- [MySQL-VERSION.src.rpm](#)

This contains the source code for all of the previous packages. It can also be used to rebuild the RPMs on other architectures (for example, Alpha or SPARC).

The suffix of RPM package names (following the [VERSION](#) value) has the following syntax:

```
.PLATFORM.CPU.rpm
```

The [PLATFORM](#) and [CPU](#) values indicate the type of system for which the package is built. [PLATFORM](#) indicates the platform and [CPU](#) indicates the processor type or family.

All packages are dynamically linked against [glibc](#) 2.3. The [PLATFORM](#) value indicates whether the package is platform independent or intended for a specific platform:

glibc23	Platform independent, should run on any Linux distribution that supports glibc 2.3
rhel3, rhel4	Red Hat Enterprise Linux 3 or 4
sles9, sles10	SuSE Linux Enterprise Server 9 or 10

In MySQL 5.1, only [glibc23](#) packages are available currently.

The [CPU](#) value indicates the processor type or family for which the package is built:

i386	x86 processor, 386 and up
i586	x86 processor, Pentium and up
x86_64	64-bit x86 processor

ia64	Itanium (IA-64) processor

To see all files in an RPM package (for example, a `MySQL-server` RPM), run a command like this:

```
shell> rpm -qpl MySQL-server-VERSION.glibc23.i386.rpm
```

To perform a standard minimal installation, install the server and client RPMs:

```
shell> rpm -i MySQL-server-VERSION.glibc23.i386.rpm
shell> rpm -i MySQL-client-VERSION.glibc23.i386.rpm
```

To install only the client programs, install just the client RPM:

```
shell> rpm -i MySQL-client-VERSION.glibc23.i386.rpm
```

RPM provides a feature to verify the integrity and authenticity of packages before installing them. If you would like to learn more about this feature, see [Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”](#).

The server RPM places data under the `/var/lib/mysql` directory. The RPM also creates a login account for a user named `mysql` (if one does not exist) to use for running the MySQL server, and creates the appropriate entries in `/etc/init.d/` to start the server automatically at boot time. (This means that if you have performed a previous installation and have made changes to its startup script, you may want to make a copy of the script so that you don't lose it when you install a newer RPM.) See [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#), for more information on how MySQL can be started automatically on system startup.

If you want to install the MySQL RPM on older Linux distributions that do not support initialization scripts in `/etc/init.d` (directly or via a symlink), you should create a symbolic link that points to the location where your initialization scripts actually are installed. For example, if that location is `/etc/rc.d/init.d`, use these commands before installing the RPM to create `/etc/init.d` as a symbolic link that points there:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

However, all current major Linux distributions should support the new directory layout that uses `/etc/init.d`, because it is required for LSB (Linux Standard Base) compliance.

If the RPM files that you install include `MySQL-server`, the `mysqld` server should be up and running after installation. You should be able to start using MySQL.

If something goes wrong, you can find more information in the binary installation section. See [Section 2.8, “Installing MySQL from tar.gz Packages on Other Unix-Like Systems”](#).

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

2.5. Installing MySQL on Mac OS X

You can install MySQL on Mac OS X 10.3.x (“Panther”) or newer using a Mac OS X binary package in PKG format instead of the binary tarball distribution. Please note that older versions of Mac OS X (for example, 10.1.x or 10.2.x) are **not** supported by this package.

The package is located inside a disk image (`.dmg`) file that you first need to mount by double-clicking its icon in the Finder. It should then mount the image and display its contents.

To obtain MySQL, see [Section 2.1.3, “How to Get MySQL”](#).

Note

Before proceeding with the installation, be sure to shut down all running MySQL server instances by either using the MySQL Manager Application (on Mac OS X Server) or via `mysqladmin shutdown` on the command line.

To actually install the MySQL PKG file, double-click on the package icon. This launches the Mac OS X Package Installer, which

guides you through the installation of MySQL.

Due to a bug in the Mac OS X package installer, you may see this error message in the destination disk selection dialog:

```
You cannot install this software on this disk. (null)
```

If this error occurs, simply click the **Go Back** button once to return to the previous screen. Then click **Continue** to advance to the destination disk selection again, and you should be able to choose the destination disk correctly. We have reported this bug to Apple and it is investigating this problem.

The Mac OS X PKG of MySQL installs itself into `/usr/local/mysql-VERSION` and also installs a symbolic link, `/usr/local/mysql`, that points to the new location. If a directory named `/usr/local/mysql` exists, it is renamed to `/usr/local/mysql.bak` first. Additionally, the installer creates the grant tables in the `mysql` database by executing `mysql_install_db`.

The installation layout is similar to that of a `tar` file binary distribution; all MySQL binaries are located in the directory `/usr/local/mysql/bin`. The MySQL socket file is created as `/tmp/mysql.sock` by default. See [Section 2.1.5, “Installation Layouts”](#).

MySQL installation requires a Mac OS X user account named `mysql`. A user account with this name should exist by default on Mac OS X 10.2 and up.

If you are running Mac OS X Server, a version of MySQL should already be installed. The following table shows the versions of MySQL that ship with Mac OS X Server versions.

Mac OS X Server Version	MySQL Version
10.2-10.2.2	3.23.51
10.2.3-10.2.6	3.23.53
10.3	4.0.14
10.3.2	4.0.16
10.4.0	4.1.10a

This manual section covers the installation of the official MySQL Mac OS X PKG only. Make sure to read Apple's help information about installing MySQL: Run the “Help View” application, select “Mac OS X Server” help, do a search for “MySQL,” and read the item entitled “Installing MySQL.”

If you previously used Marc Liyanage's MySQL packages for Mac OS X from <http://www.entropy.ch>, you can simply follow the update instructions for packages using the binary installation layout as given on his pages.

If you are upgrading from Marc's 3.23.x versions or from the Mac OS X Server version of MySQL to the official MySQL PKG, you also need to convert the existing MySQL privilege tables to the current format, because some new security privileges have been added. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

If you want MySQL to start automatically during system startup, you also need to install the MySQL Startup Item. It is part of the Mac OS X installation disk images as a separate installation package. Simply double-click the `MySQLStartupItem.pkg` icon and follow the instructions to install it. The Startup Item need be installed only once. There is no need to install it each time you upgrade the MySQL package later.

The Startup Item for MySQL is installed into `/Library/StartupItems/MySQLCOM`. (Before MySQL 4.1.2, the location was `/Library/StartupItems/MySQL`, but that collided with the MySQL Startup Item installed by Mac OS X Server.) Startup Item installation adds a variable `MYSQLCOM=-YES-` to the system configuration file `/etc/hostconfig`. If you want to disable the automatic startup of MySQL, simply change this variable to `MYSQLCOM=-NO-`.

On Mac OS X Server, the default MySQL installation uses the variable `MYSQL` in the `/etc/hostconfig` file. The MySQL AB Startup Item installer disables this variable by setting it to `MYSQL=-NO-`. This avoids boot time conflicts with the `MYSQLCOM` variable used by the MySQL AB Startup Item. However, it does not shut down a running MySQL server. You should do that yourself.

After the installation, you can start up MySQL by running the following commands in a terminal window. You must have administrator privileges to perform this task.

If you have installed the Startup Item, use this command:

```
shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM start
(Enter your password, if necessary)
(Press Control-D or enter "exit" to exit the shell)
```

If you don't use the Startup Item, enter the following command sequence:

```
shell> cd /usr/local/mysql
shell> sudo ./bin/mysqld_safe
(Enter your password, if necessary)
(Press Control-Z)
shell> bg
(Press Control-D or enter "exit" to exit the shell)
```

You should be able to connect to the MySQL server, for example, by running `/usr/local/mysql/bin/mysql`.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

You might want to add aliases to your shell's resource file to make it easier to access commonly used programs such as `mysql` and `mysqladmin` from the command line. The syntax for `bash` is:

```
alias mysql=/usr/local/mysql/bin/mysql
alias mysqladmin=/usr/local/mysql/bin/mysqladmin
```

For `tcsh`, use:

```
alias mysql /usr/local/mysql/bin/mysql
alias mysqladmin /usr/local/mysql/bin/mysqladmin
```

Even better, add `/usr/local/mysql/bin` to your `PATH` environment variable. You can do this by modifying the appropriate startup file for your shell. For more information, see [Section 4.2, “Invoking MySQL Programs”](#).

If you are upgrading an existing installation, note that installing a new MySQL PKG does not remove the directory of an older installation. Unfortunately, the Mac OS X Installer does not yet offer the functionality required to properly upgrade previously installed packages.

To use your existing databases with the new installation, you'll need to copy the contents of the old data directory to the new data directory. Make sure that neither the old server nor the new one is running when you do this. After you have copied over the MySQL database files from the previous installation and have successfully started the new server, you should consider removing the old installation files to save disk space. Additionally, you should also remove older versions of the Package Receipt directories located in `/Library/Receipts/mysql-VERSION.pkg`.

2.6. Installing MySQL on Solaris

If you install MySQL using a binary tarball distribution on Solaris, you may run into trouble even before you get the MySQL distribution unpacked, as the Solaris `tar` cannot handle long filenames. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution. You can find a precompiled copy for Solaris at <http://dev.mysql.com/downloads/os-solaris.html>.

You can install MySQL on Solaris using a binary package in PKG format instead of the binary tarball distribution. Before installing using the binary PKG format, you should create the `mysql` user and group, for example:

```
groupadd mysql
useradd -g mysql mysql
```

Some basic PKG-handling commands follow:

- To add a package:

```
pkgadd -d package_name.pkg
```

- To remove a package:

```
pkgrm package_name
```

- To get a full list of installed packages:

```
pkginfo
```

- To get detailed information for a package:

```
pkginfo -l package_name
```

- To list the files belonging to a package:

```
pkgchk -v package_name
```

- To get packaging information for an arbitrary file:

```
pkgchk -l -p file_name
```

For additional information about installing MySQL on Solaris, see [Section 2.13.3, “Solaris Notes”](#).

2.7. Installing MySQL on NetWare

Porting MySQL to NetWare was an effort spearheaded by Novell. Novell customers should be pleased to note that NetWare 6.5 ships with bundled MySQL binaries, complete with an automatic commercial use license for all servers running that version of NetWare.

MySQL for NetWare is compiled using a combination of Metrowerks CodeWarrior for NetWare and special cross-compilation versions of the GNU autotools.

The latest binary packages for NetWare can be obtained at <http://dev.mysql.com/downloads/>. See [Section 2.1.3, “How to Get MySQL”](#).

To host MySQL, the NetWare server must meet these requirements:

- The latest Support Pack of [NetWare 6.5](#) must be installed.
- The system must meet Novell's minimum requirements to run the respective version of NetWare.
- MySQL data and the program binaries must be installed on an NSS volume; traditional volumes are not supported.

To install MySQL for NetWare, use the following procedure:

1. If you are upgrading from a prior installation, stop the MySQL server. This is done from the server console, using the following command:

```
SERVER: mysqladmin -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

2. Log on to the target server from a client machine with access to the location where you are installing MySQL.
3. Extract the binary package Zip file onto the server. Be sure to allow the paths in the Zip file to be used. It is safe to simply extract the file to `SYS:\`.

If you are upgrading from a prior installation, you may need to copy the data directory (for example, `SYS:MYSQL\DATA`), as well as `my.cnf`, if you have customized it. You can then delete the old copy of MySQL.

4. You might want to rename the directory to something more consistent and easy to use. The examples in this manual use `SYS:MYSQL` to refer to the installation directory.

Note that MySQL installation on NetWare does not detect if a version of MySQL is already installed outside the NetWare release. Therefore, if you have installed the latest MySQL version from the Web (for example, MySQL 4.1 or later) in `SYS:\MYSQL`, you must rename the folder before upgrading the NetWare server; otherwise, files in `SYS:\MYSQL` are overwritten by the MySQL version present in NetWare Support Pack.

5. At the server console, add a search path for the directory containing the MySQL NLMs. For example:

```
SERVER:  SEARCH ADD SYS:MYSQL\BIN
```

6. Initialize the data directory and the grant tables, if necessary, by executing `mysql_install_db` at the server console.
7. Start the MySQL server using `mysqld_safe` at the server console.
8. To finish the installation, you should also add the following commands to `autoexec.ncf`. For example, if your MySQL installation is in `SYS:MYSQL` and you want MySQL to start automatically, you could add these lines:

```
#Starts the MySQL 5.1.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE
```

If you are running MySQL on NetWare 6.0, we strongly suggest that you use the `--skip-external-locking` option on the command line:

```
#Starts the MySQL 5.1.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --skip-external-locking
```

It is also necessary to use `CHECK TABLE` and `REPAIR TABLE` instead of `myisamchk`, because `myisamchk` makes use of external locking. External locking is known to have problems on NetWare 6.0; the problem has been eliminated in NetWare 6.5. Note that the use of MySQL on Netware 6.0 is not officially supported.

`mysqld_safe` on NetWare provides a screen presence. When you unload (shut down) the `mysqld_safe` NLM, the screen does not go away by default. Instead, it prompts for user input:

```
*<NLM has terminated; Press any key to close the screen>*
```

If you want NetWare to close the screen automatically instead, use the `--autoclose` option to `mysqld_safe`. For example:

```
#Starts the MySQL 5.1.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --autoclose
```

The behavior of `mysqld_safe` on NetWare is described further in [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#).

9. When installing MySQL, either for the first time or upgrading from a previous version, download and install the latest and appropriate Perl module and PHP extensions for NetWare:
 - Perl: <http://forge.novell.com/modules/xfcontent/downloads.php/perl/Modules/>
 - PHP: <http://forge.novell.com/modules/xfcontent/downloads.php/php/Modules/>

If there was an existing installation of MySQL on the NetWare server, be sure to check for existing MySQL startup commands in `autoexec.ncf`, and edit or delete them as necessary.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

2.8. Installing MySQL from `tar.gz` Packages on Other Unix-Like Systems

This section covers the installation of MySQL binary distributions that are provided for various platforms in the form of compressed `tar` files (files with a `.tar.gz` extension). See [Section 2.1.2.5, “MySQL Binaries Compiled by MySQL AB”](#), for a detailed list.

To obtain MySQL, see [Section 2.1.3, “How to Get MySQL”](#).

MySQL `tar` file binary distributions have names of the form `mysql-VERSION-OS.tar.gz`, where `VERSION` is a number (for example, `5.1.21-beta`), and `OS` indicates the type of operating system for which the distribution is intended (for example, `pc-linux-i686`).

In addition to these generic packages, we also offer binaries in platform-specific package formats for selected platforms. See [Section 2.2, “Standard MySQL Installation Using a Binary Distribution”](#), for more information on how to install these.

You need the following tools to install a MySQL `tar` file binary distribution:

- GNU `gunzip` to uncompress the distribution.
- A reasonable `tar` to unpack the distribution. GNU `tar` is known to work. Some operating systems come with a preinstalled version of `tar` that is known to have problems. For example, the `tar` provided with early versions of Mac OS X, SunOS 4.x and Solaris 8 and earlier are known to have problems with long filenames. On Mac OS X, you can use the preinstalled `gnutar` program. On other systems with a deficient `tar`, you should install GNU `tar` first.

If you run into problems and need to file a bug report, please use the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

The basic commands that you must execute to install and use a MySQL binary distribution are:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> cd /usr/local
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql data
shell> bin/mysqld_safe --user=mysql &
```

Note

This procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to [Section 2.10, “Post-Installation Setup and Testing”](#).

A more detailed version of the preceding description for installing a binary distribution follows:

1. Add a login user and group for `mysqld` to run as:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

These commands add the `mysql` group and the `mysql` user. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following steps.

2. Pick the directory under which you want to unpack the distribution and change location into it. In the following example, we unpack the distribution under `/usr/local`. (The instructions, therefore, assume that you have permission to create files and directories in `/usr/local`. If that directory is protected, you must perform the installation as `root`.)

```
shell> cd /usr/local
```

- Obtain a distribution file using the instructions in [Section 2.1.3, “How to Get MySQL”](#). For a given release, binary distributions for all platforms are built from the same MySQL source distribution.
- Unpack the distribution, which creates the installation directory. Then create a symbolic link to that directory:

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

The `tar` command creates a directory named `mysql-VERSION-OS`. The `ln` command makes a symbolic link to that directory. This lets you refer more easily to the installation directory as `/usr/local/mysql`.

With GNU `tar`, no separate invocation of `gunzip` is necessary. You can replace the first line with the following alternative command to uncompress and extract the distribution:

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
```

- Change location into the installation directory:

```
shell> cd mysql
```

You will find several files and subdirectories in the `mysql` directory. The most important for installation purposes are the `bin` and `scripts` subdirectories:

- The `bin` directory contains client programs and the server. You should add the full pathname of this directory to your `PATH` environment variable so that your shell finds the MySQL programs properly. See [Section 2.14, “Environment Variables”](#).
 - The `scripts` directory contains the `mysql_install_db` script used to initialize the `mysql` database containing the grant tables that store the server access permissions.
- Ensure that the distribution contents are accessible to `mysql`. If you unpacked the distribution as `mysql`, no further action is required. If you unpacked the distribution as `root`, its contents will be owned by `root`. Change its ownership to `mysql` by executing the following commands as `root` in the installation directory:

```
shell> chown -R mysql .
shell> chgrp -R mysql .
```

The first command changes the owner attribute of the files to the `mysql` user. The second changes the group attribute to the `mysql` group.

- If you have not installed MySQL before, you must create the MySQL data directory and initialize the grant tables:

```
shell> scripts/mysql_install_db --user=mysql
```

If you run the command as `root`, include the `--user` option as shown. If you run the command while logged in as that user, you can omit the `--user` option.

The command should create the data directory and its contents with `mysql` as the owner.

After creating or updating the grant tables, you need to restart the server manually.

- Most of the MySQL installation can be owned by `root` if you like. The exception is that the data directory must be owned by `mysql`. To accomplish this, run the following commands as `root` in the installation directory:

```
shell> chown -R root .
shell> chown -R mysql data
```

- If you want MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `support-files/mysql.server` script itself and in [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).
- You can set up new accounts using the `bin/mysql_setpermission` script if you install the `DBI` and `DBD:mysql` Perl modules. See [Section 7.21, “mysql_setpermission — Interactively Set Permissions in Grant Tables”](#). For Perl module installation instructions, see [Section 2.15, “Perl Installation Notes”](#).

11. If you would like to use `mysqlaccess` and have the MySQL distribution in some non-standard location, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the `bin/mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL = '/usr/local/bin/mysql'; # path to mysql executable
```

Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, a **Broken pipe** error will occur when you run `mysqlaccess`.

After everything has been unpacked and installed, you should test your distribution. To start the MySQL server, use the following command:

```
shell> bin/mysqld_safe --user=mysql &
```

If you run the command as `root`, you must use the `--user` option as shown. The value of the option is the name of the login account that you created in the first step to use for running the server. If you run the command while logged in as `mysql`, you can omit the `--user` option.

If the command fails immediately and prints `mysqld ended`, you can find some information in the `host_name.err` file in the data directory.

More information about `mysqld_safe` is given in [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#).

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

2.9. MySQL Installation Using a Source Distribution

Before you proceed with an installation from source, first check whether our binary is available for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options.

To obtain a source distribution for MySQL, [Section 2.1.3, “How to Get MySQL”](#). If you want to build MySQL from source on Windows, see [Section 2.9.6, “Installing MySQL from Source on Windows”](#).

MySQL source distributions are provided as compressed `tar` archives and have names of the form `mysql-VERSION.tar.gz`, where `VERSION` is a number like `5.1.21-beta`.

You need the following tools to build and install MySQL from source:

- GNU `gunzip` to uncompress the distribution.
- A reasonable `tar` to unpack the distribution. GNU `tar` is known to work. Some operating systems come with a preinstalled version of `tar` that is known to have problems. For example, the `tar` provided with early versions of Mac OS X, SunOS 4.x and Solaris 8 and earlier are known to have problems with long filenames. On Mac OS X, you can use the preinstalled `gnutar` program. On other systems with a deficient `tar`, you should install GNU `tar` first.
- A working ANSI C++ compiler. `gcc` 2.95.2 or later, `egcs` 1.0.2 or later or `egcs` 2.91.66, SGI C++, and SunPro C++ are some of the compilers that are known to work. `libg++` is not needed when using `gcc`. `gcc` 2.7.x has a bug that makes it impossible to compile some perfectly legal C++ files, such as `sql/sql_base.cc`. If you have only `gcc` 2.7.x, you must upgrade your `gcc` to be able to compile MySQL. `gcc` 2.8.1 is also known to have problems on some platforms, so it should be avoided if a new compiler exists for the platform.

`gcc` 2.95.2 or later is recommended when compiling MySQL 3.23.x.
- A good `make` program. GNU `make` is always recommended and is sometimes required. If you have problems, we recommend GNU `make` 3.75 or newer.

If you are using a version of `gcc` recent enough to understand the `-fno-exceptions` option, it is *very important* that you use this option. Otherwise, you may compile a binary that crashes randomly. We also recommend that you use `-felide-constructors`

and `-fno-rtti` along with `-fno-exceptions`. When in doubt, do the following:

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors \
-fno-exceptions -fno-rtti" ./configure \
--prefix=/usr/local/mysql --enable-assembly \
--with-mysqld-ldflags=-all-static
```

On most systems, this gives you a fast and stable binary.

If you run into problems and need to file a bug report, please use the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

2.9.1. Source Installation Overview

The basic commands that you must execute to install a MySQL source distribution are:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> gunzip < mysql-VERSION.tar.gz | tar -xvf -
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> cd /usr/local/mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> bin/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql var
shell> bin/mysqld_safe --user=mysql &
```

If you start from a source RPM, do the following:

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

This makes a binary RPM that you can install. For older versions of RPM, you may have to replace the command `rpmbuild` with `rpm` instead.

Note

This procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to [Section 2.10, “Post-Installation Setup and Testing”](#), for post-installation setup and testing.

A more detailed version of the preceding description for installing MySQL from a source distribution follows:

1. Add a login user and group for `mysqld` to run as:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

These commands add the `mysql` group and the `mysql` user. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following steps.

2. Perform the following steps as the `mysql` user, except as noted.
3. Pick the directory under which you want to unpack the distribution and change location into it.
4. Obtain a distribution file using the instructions in [Section 2.1.3, “How to Get MySQL”](#).
5. Unpack the distribution into the current directory:

```
shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
```

This command creates a directory named `mysql-VERSION`.

With GNU `tar`, no separate invocation of `gunzip` is necessary. You can use the following alternative command to uncompress and extract the distribution:

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
```

6. Change location into the top-level directory of the unpacked distribution:

```
shell> cd mysql-VERSION
```

Note that currently you must configure and build MySQL from this top-level directory. You cannot build it in a different directory.

7. Configure the release and compile everything:

```
shell> ./configure --prefix=/usr/local/mysql
shell> make
```

When you run `configure`, you might want to specify other options. Run `./configure --help` for a list of options. [Section 2.9.2, “Typical configure Options”](#), discusses some of the more useful options.

If `configure` fails and you are going to send mail to a MySQL mailing list to ask for assistance, please include any lines from `config.log` that you think can help solve the problem. Also include the last couple of lines of output from `configure`. To file a bug report, please use the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

If the compile fails, see [Section 2.9.4, “Dealing with Problems Compiling MySQL”](#), for help.

8. Install the distribution:

```
shell> make install
```

You might need to run this command as `root`.

If you want to set up an option file, use one of those present in the `support-files` directory as a template. For example:

```
shell> cp support-files/my-medium.cnf /etc/my.cnf
```

You might need to run this command as `root`.

If you want to configure support for `InnoDB` tables, you should edit the `/etc/my.cnf` file, remove the `#` character before the option lines that start with `innodb_...`, and modify the option values to be what you want. See [Section 4.3.2, “Using Option Files”](#), and [Section 13.5.3, “InnoDB Configuration”](#).

9. Change location into the installation directory:

```
shell> cd /usr/local/mysql
```

10. If you ran the `make install` command as `root`, the installed files will be owned by `root`. Ensure that the installation is accessible to `mysql` by executing the following commands as `root` in the installation directory:

```
shell> chown -R mysql .
shell> chgrp -R mysql .
```

The first command changes the owner attribute of the files to the `mysql` user. The second changes the group attribute to the `mysql` group.

11. If you have not installed MySQL before, you must create the MySQL data directory and initialize the grant tables:

```
shell> bin/mysql_install_db --user=mysql
```

If you run the command as `root`, include the `--user` option as shown. If you run the command while logged in as `mysql`, you can omit the `--user` option.

The command should create the data directory and its contents with `mysql` as the owner.

After using `mysql_install_db` to create the grant tables for MySQL, you must restart the server manually. The `mysqld_safe` command to do this is shown in a later step.

12. Most of the MySQL installation can be owned by `root` if you like. The exception is that the data directory must be owned by `mysql`. To accomplish this, run the following commands as `root` in the installation directory:

```
shell> chown -R root .
shell> chown -R mysql var
```

13. If you want MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `support-files/mysql.server` script itself; see also [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).
14. You can set up new accounts using the `bin/mysql_setpermission` script if you install the `DBI` and `DBD::mysql` Perl modules. See [Section 7.21, “mysql_setpermission — Interactively Set Permissions in Grant Tables”](#). For Perl module installation instructions, see [Section 2.15, “Perl Installation Notes”](#).

After everything has been installed, you should test your distribution. To start the MySQL server, use the following command:

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

If you run the command as `root`, you should use the `--user` option as shown. The value of the option is the name of the login account that you created in the first step to use for running the server. If you run the command while logged in as that user, you can omit the `--user` option.

If the command fails immediately and prints `mysqld ended`, you can find some information in the `host_name.err` file in the data directory.

More information about `mysqld_safe` is given in [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#).

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

2.9.2. Typical `configure` Options

The `configure` script gives you a great deal of control over how you configure a MySQL source distribution. Typically you do this using options on the `configure` command line. You can also affect `configure` using certain environment variables. See [Section 2.14, “Environment Variables”](#). For a full list of options supported by `configure`, run this command:

```
shell> ./configure --help
```

Some of the `configure` options available are described here:

- To compile just the MySQL client libraries and client programs and not the server, use the `--without-server` option:

```
shell> ./configure --without-server
```

If you have no C++ compiler, some client programs such as `mysql` cannot be compiled because they require C++. In this case, you can remove the code in `configure` that tests for the C++ compiler and then run `./configure` with the `--without-server` option. The compile step should still try to build all clients, but you can ignore any warnings about files such as `mysql.cc`. (If `make` stops, try `make -k` to tell it to continue with the rest of the build even if errors occur.)

- If you want to build the embedded MySQL library (`libmysqld.a`), use the `--with-embedded-server` option.
- If you don't want your log files and database directories located under `/usr/local/var`, use a `configure` command something like one of these:

```
shell> ./configure --prefix=/usr/local/mysql
shell> ./configure --prefix=/usr/local \
    --localstatedir=/usr/local/mysql/data
```

The first command changes the installation prefix so that everything is installed under `/usr/local/mysql` rather than the default of `/usr/local`. The second command preserves the default installation prefix, but overrides the default location for database directories (normally `/usr/local/var`) and changes it to `/usr/local/mysql/data`.

You can also specify the installation directory and data directory locations at server startup time by using the `--basedir` and `--datadir` options. These can be given on the command line or in an MySQL option file, although it is more common to use an option file. See [Section 4.3.2, “Using Option Files”](#).

- If you are using Unix and you want the MySQL socket file location to be somewhere other than the default location (normally in the directory `/tmp` or `/var/run`), use a `configure` command like this:

```
shell> ./configure \
        --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

The socket filename must be an absolute pathname. You can also change the location of `mysql.sock` at server startup by using a MySQL option file. See [Section B.1.4.5, “How to Protect or Change the MySQL Unix Socket File”](#).

- If you want to compile statically linked programs (for example, to make a binary distribution, to get better performance, or to work around problems with some Red Hat Linux distributions), run `configure` like this:

```
shell> ./configure --with-client-ldflags=-all-static \
        --with-mysqld-ldflags=-all-static
```

- If you are using `gcc` and don't have `libg++` or `libstdc++` installed, you can tell `configure` to use `gcc` as your C++ compiler:

```
shell> CC=gcc CXX=gcc ./configure
```

When you use `gcc` as your C++ compiler, it does not attempt to link in `libg++` or `libstdc++`. This may be a good thing to do even if you have those libraries installed. Some versions of them have caused strange problems for MySQL users in the past.

The following list indicates some compilers and environment variable settings that are commonly used with each one.

- `gcc` 2.7.2:

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors"
```

- `egcs` 1.0.3a:

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors \
-fno-exceptions -fno-rtti"
```

- `gcc` 2.95.2:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti"
```

- `pgcc` 2.90.29 or newer:

```
CFLAGS="-O3 -mpentiumpro -mstack-align-double" CXX=gcc \
CXXFLAGS="-O3 -mpentiumpro -mstack-align-double \
-felide-constructors -fno-exceptions -fno-rtti"
```

In most cases, you can get a reasonably optimized MySQL binary by using the options from the preceding list and adding the following options to the `configure` line:

```
--prefix=/usr/local/mysql --enable-asm \
--with-mysqld-ldflags=-all-static
```

The full `configure` line would, in other words, be something like the following for all recent `gcc` versions:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti" ./configure \
```

```
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

The binaries we provide on the MySQL Web site at <http://dev.mysql.com/downloads/> are all compiled with full optimization and should be perfect for most users. See [Section 2.1.2.5, “MySQL Binaries Compiled by MySQL AB”](#). There are some configuration settings you can tweak to build an even faster binary, but these are only for advanced users. See [Section 6.5.6, “How Compiling and Linking Affects the Speed of MySQL”](#).

If the build fails and produces errors about your compiler or linker not being able to create the shared library `libmysqlclient.so.N` (where *N* is a version number), you can work around this problem by giving the `--disable-shared` option to `configure`. In this case, `configure` does not build a shared `libmysqlclient.so.N` library.

- By default, MySQL uses the `latin1` (cp1252 West European) character set. To change the default set, use the `--with-charset` option:

```
shell> ./configure --with-charset=CHARSET
```

`CHARSET` may be one of `binary`, `armscii8`, `ascii`, `big5`, `cp1250`, `cp1251`, `cp1256`, `cp1257`, `cp850`, `cp852`, `cp866`, `cp932`, `dec8`, `eucjpms`, `euckr`, `gb2312`, `gbk`, `geostd8`, `greek`, `hebrew`, `hp8`, `keybcs2`, `koi8r`, `koi8u`, `latin1`, `latin2`, `latin5`, `latin7`, `macce`, `macroman`, `sjis`, `swe7`, `tis620`, `ucs2`, `ujis`, `utf8`. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#). (Additional character sets might be available. Check the output from `./configure --help` for the current list.)

The default collation may also be specified. MySQL uses the `latin1_swedish_ci` collation by default. To change this, use the `--with-collation` option:

```
shell> ./configure --with-collation=COLLATION
```

To change both the character set and the collation, use both the `--with-charset` and `--with-collation` options. The collation must be a legal collation for the character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.)

Warning

If you change character sets after having created any tables, you must run `myisamchk -r -q --set-collation=collation_name` on every *MyISAM* table. Your indexes may be sorted incorrectly otherwise. This can happen if you install MySQL, create some tables, and then reconfigure MySQL to use a different character set and reinstall it.

With the `configure` option `--with-extra-charsets=LIST`, you can define which additional character sets should be compiled into the server. `LIST` is one of the following:

- A list of character set names separated by spaces
- `complex` to include all character sets that can't be dynamically loaded
- `all` to include all character sets into the binaries

Clients that want to convert characters between the server and the client should use the `SET NAMES` statement. See [Section 12.5.3, “SET Syntax”](#), and [Section 9.4, “Connection Character Sets and Collations”](#).

- To configure MySQL with debugging code, use the `--with-debug` option:

```
shell> ./configure --with-debug
```

This causes a safe memory allocator to be included that can find some errors and that provides output about what is happening. See [MySQL Internals: Porting](#).

As of MySQL 5.1.12, using `--with-debug` to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

- If your client programs are using threads, you must compile a thread-safe version of the MySQL client library with the `-`

`--enable-thread-safe-client` configure option. This creates a `libmysqlclient_r` library with which you should link your threaded applications. See [Section 25.2.16](#), “How to Make a Threaded Client”.

- It is possible to build MySQL with large table support using the `--with-big-tables` option.

This option causes the variables that store table row counts to be declared as `unsigned long long` rather than `unsigned long`. This enables tables to hold up to approximately $1.844\text{E}+19$ ($(2^{32})^2$) rows rather than 2^{32} ($\sim 4.295\text{E}+09$) rows. Previously it was necessary to pass `-DBIG_TABLES` to the compiler manually in order to enable this feature.

- Run `configure` with the `--disable-grant-options` option to cause the `--bootstrap`, `--skip-grant-tables`, and `--init-file` options for `mysqld` to be disabled. For Windows, the `configure.js` script recognizes the `DISABLE_GRANT_OPTIONS` flag, which has the same effect. The capability is available as of MySQL 5.1.15.
- See [Section 2.13](#), “Operating System-Specific Notes”, for options that pertain to particular operating systems.
- See [Section 5.8.7.2](#), “Using SSL Connections”, for options that pertain to configuring MySQL to support secure (encrypted) connections.
- Several `configure` options apply to plugin selection and building. You can build a plugin as static (compiled into the server) or dynamic (built as a dynamic library that must be installed using the `INSTALL PLUGIN` statement before it can be used). Some plugins might not support static or dynamic build.

`configure --help` shows the following information pertaining to plugins:

- The plugin-related options
- The names of all available plugins
- For each plugin, a description of its purpose, which build types it supports (static or dynamic), and which plugin groups it is a part of.

The following `configure` options are used to select or disable plugins:

```
--with-plugins=PLUGIN[,PLUGIN]...
--with-plugins=GROUP
--with-plugin=PLUGIN
--without-plugin=PLUGIN
```

`PLUGIN` is an individual plugin name such as `csv` or `archive`.

As shorthand, `GROUP` is a configuration group name such as `none` (select no plugins) or `all` (select all plugins).

`--with-plugins` can take a list of one or more plugin names separated by commas, or a plugin group name. The named plugins are configured to be built as static plugins.

`--with-plugin=PLUGIN` configures the given plugin to be built as a static plugin.

`--without-plugin=PLUGIN` disables the given plugin from being built.

If a plugin is named both with a `--with` and `--without` option, the result is undefined.

For any plugin that is not explicitly selected or disabled, it is selected to be built dynamically if it supports dynamic build, and not built if it does not support dynamic build. (Thus, in the case that no plugin options are given, all plugins that support dynamic build are selected to be built as dynamic plugins. Plugins that do not support dynamic build are not built.)

2.9.3. Installing from the Development Source Tree

Caution

You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL up and running on your system, you should use a standard release distribution (either a binary or source distribution).

To obtain our most recent development source tree, first download and install the BitKeeper free client if you do not have it. The client can be obtained from <http://www.bitmover.com/bk-client2.0.shar>. Note that you will need `gcc` and `make` to build the BitKeeper free client, and `patch` and `tar` to use the BitKeeper free client. *Note that old 1.1 versions of the BitKeeper free client will not work!*

To install the BitKeeper client on Unix, use these commands:

```
shell> /bin/sh bk-client2.0.shar
shell> cd bk-client2.0
shell> make
```

If you get a `cc: command not found` error, invoke this command before running `make`:

```
shell> make CC=gcc
```

The step above will create the utility `bkf`, which is the BitKeeper free client. You may use the BitKeeper free client in the same way as the main client. For more information on `bkf`, use:

```
shell> bkf --help
```

To install the BitKeeper client on Windows, use these instructions:

1. Download and install Cygwin from <http://cygwin.com>.
2. Make sure `patch`, `tar`, `gcc` and `make` have been installed under Cygwin. You can test this by issuing `which gcc` for each command. If a required tool is not installed, run Cygwin's package manager, select the required tools and install them.
3. For the installation of the BitKeeper free client, use the same installations as given for Unix-like systems above.

The BitKeeper free client is shipped with its source code. The only documentation available for the free client is the source code itself.

After you have installed the BitKeeper client, you can access the MySQL development source tree:

1. Change location to the directory you want to work from, and then use the following command to make a local copy of the MySQL 5.1 branch:

```
shell> bkf clone bk://mysql.bkbits.net/mysql-5.1 mysql-5.1
```

In the preceding example, the source tree is set up in the `mysql-5.1/` subdirectory of your current directory.

The initial download of the source tree may take a while, depending on the speed of your connection. Please be patient.

2. You need GNU `make`, `autoconf` 2.58 (or newer), `automake` 1.8.1, `libtool` 1.5, and `m4` to run the next set of commands. Even though many operating systems come with their own implementation of `make`, chances are high that the compilation fails with strange error messages. Therefore, it is highly recommended that you use GNU `make` (sometimes named `gmake`) instead.

Fortunately, a large number of operating systems ship with the GNU toolchain preinstalled or supply installable packages of these. In any case, they can also be downloaded from the following locations:

- <http://www.gnu.org/software/autoconf/>
- <http://www.gnu.org/software/automake/>
- <http://www.gnu.org/software/libtool/>
- <http://www.gnu.org/software/m4/>
- <http://www.gnu.org/software/make/>

To configure MySQL 5.1, you also need GNU `bison`. You should use the latest version of bison where possible. Version 1.75 and version 2.1 are known to work. There have been reported problems with `bison` 1.875. If you experience problems, upgrade to a later, rather than earlier, version. Versions of `bison` older than 1.75 may report this error:

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
```

Note: The maximum table size is not actually exceeded; the error is caused by bugs in older versions of `bison`.

The following example shows the typical commands required to configure a source tree. The first `cd` command changes location into the top-level directory of the tree; replace `mysql-5.1` with the appropriate directory name. The second line (for `storage/innobase`) is needed only before MySQL 5.1.12.

```
shell> cd mysql-5.1
shell> (cd storage/innobase; autoreconf --force --install)
shell> autoreconf --force --install
shell> ./configure # Add your favorite options here
shell> make
```

Or you can use `BUILD/autorun.sh` as a shortcut for the following sequence of commands:

```
shell> aclocal; autoheader
shell> libtoolize --automake --force
shell> automake --force --add-missing; autoconf
shell> (cd storage/innobase; aclocal; autoheader; autoconf; automake)
```

The command line that changes directory into the `storage/innobase` directory is used to configure the `InnoDB` storage engine. You can omit this line if you do not require `InnoDB` support.

Note

Beginning with MySQL 5.1, code specific to storage engines has been moved under a `storage` directory. For example, `InnoDB` code is now found in `storage/innobase` and `NDBCluster` code is in `storage/ndb`.

If you get some strange errors during this stage, verify that you really have `libtool` installed.

A collection of our standard configuration scripts is located in the `BUILD/` subdirectory. You may find it more convenient to use the `BUILD/compile-pentium-debug` script than the preceding set of shell commands. To compile on a different architecture, modify the script by removing flags that are Pentium-specific.

- When the build is done, run `make install`. Be careful with this on a production machine; the command may overwrite your live release installation. If you have another installation of MySQL, we recommend that you run `./configure` with different values for the `--prefix`, `--with-tcp-port`, and `--unix-socket-path` options than those used for your production server.
- Play hard with your new installation and try to make the new features crash. Start by running `make test`. See [Section 28.1.2, “MySQL Test Suite”](#).
- If you have gotten to the `make` stage, but the distribution does not compile, please enter the problem into our bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#). If you have installed the latest versions of the required GNU tools, and they crash trying to process our configuration files, please report that also. However, if you execute `aclocal` and get a `command not found` error or a similar problem, do not report it. Instead, make sure that all the necessary tools are installed and that your `PATH` variable is set correctly so that your shell can find them.
- After initially copying the repository with `bkf` to obtain the source tree, you should use `pull` option to periodically update your local copy. To do this any time after you have set up the repository, use this command:

```
shell> bkf pull
```

- You can examine the changeset comments for the tree by using the `changes` option to `bkf`:

```
shell> bkf changes
```

To get a list of the changes that would be applied with the next `bkf pull`:

```
shell> bkf changes -R
```

To obtain a patch file for a specific changeset (`CSETID`), use:

```
shell> bkf changes -vvrCSETID
```

If you see diffs or code that you have a question about, do not hesitate to send email to the MySQL [internals](#) mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#). Also, if you think you have a better idea on how to do something, send an email message

to the list with a patch.

You can also browse changesets, comments, and source code online. To browse this information for MySQL 5.1, go to <http://mysql.bkbits.net:8080/mysql-5.1>.

2.9.4. Dealing with Problems Compiling MySQL

All MySQL programs compile cleanly for us with no warnings on Solaris or Linux using `gcc`. On other systems, warnings may occur due to differences in system include files. See [Section 2.9.5, “MIT-pthreads Notes”](#), for warnings that may occur when using MIT-pthreads. For other problems, check the following list.

The solution to many problems involves reconfiguring. If you do need to reconfigure, take note of the following:

- If `configure` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `config.cache`. When `configure` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `configure`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old configuration information or object files from being used, run these commands before re-running `configure`:

```
shell> rm config.cache
shell> make clean
```

Alternatively, you can run `make distclean`.

The following list describes some of the problems when compiling MySQL that have been found to occur most often:

- If you get errors such as the ones shown here when compiling `sql_yacc.cc`, you probably have run out of memory or swap space:

```
Internal compiler error: program cc1plus got fatal signal 11
Out of virtual memory
Virtual memory exhausted
```

The problem is that `gcc` requires a huge amount of memory to compile `sql_yacc.cc` with inline functions. Try running `configure` with the `--with-low-memory` option:

```
shell> ./configure --with-low-memory
```

This option causes `-fno-inline` to be added to the compile line if you are using `gcc` and `-O0` if you are using something else. You should try the `--with-low-memory` option even if you have so much memory and swap space that you think you can't possibly have run out. This problem has been observed to occur even on systems with generous hardware configurations, and the `--with-low-memory` option usually fixes it.

- By default, `configure` picks `c++` as the compiler name and GNU `c++` links with `-lg++`. If you are using `gcc`, that behavior can cause problems during configuration such as this:

```
configure: error: installation or configuration problem:
C++ compiler cannot create executables.
```

You might also observe problems during compilation related to `g++`, `libg++`, or `libstdc++`.

One cause of these problems is that you may not have `g++`, or you may have `g++` but not `libg++`, or `libstdc++`. Take a look at the `config.log` file. It should contain the exact reason why your C++ compiler didn't work. To work around these problems, you can use `gcc` as your C++ compiler. Try setting the environment variable `CXX` to `"gcc -O3"`. For example:

```
shell> CXX="gcc -O3" ./configure
```

This works because `gcc` compiles C++ source files as well as `g++` does, but does not link in `libg++` or `libstdc++` by default.

Another way to fix these problems is to install `g++`, `libg++`, and `libstdc++`. However, we recommend that you not use `libg++` or `libstdc++` with MySQL because this only increases the binary size of `mysqld` without providing any benefits. Some versions of these libraries have also caused strange problems for MySQL users in the past.

- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

```
making all in mit-pthreads
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
```

Or:

```
make: file `Makefile' line 18: Must be a separator (:
```

Or:

```
pthread.h: No such file or directory
```

Solaris and FreeBSD are known to have troublesome `make` programs.

GNU `make` 3.75 is known to work.

- If you want to define flags to be used by your C or C++ compilers, do so by adding the flags to the `CFLAGS` and `CXXFLAGS` environment variables. You can also specify the compiler names this way using `CC` and `CXX`. For example:

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

See [Section 2.1.2.5, “MySQL Binaries Compiled by MySQL AB”](#), for a list of flag definitions that have been found to be useful on various systems.

- If you get errors such as those shown here when compiling `mysqld`, `configure` did not correctly detect the type of the last argument to `accept()`, `getsockname()`, or `getpeername()`:

```
cxx: Error: mysqld.cc, line 645: In this statement, the referenced
      type of the pointer value 'length' is 'unsigned long',
      which is not compatible with 'int'.
new_sock = accept(sock, (struct sockaddr *)&cAddr, &length);
```

To fix this, edit the `config.h` file (which is generated by `configure`). Look for these lines:

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

Change `XXX` to `size_t` or `int`, depending on your operating system. (You must do this each time you run `configure` because `configure` regenerates `config.h`.)

- The `sql_yacc.cc` file is generated from `sql_yacc.yy`. Normally, the build process does not need to create `sql_yacc.cc` because MySQL comes with a pre-generated copy. However, if you do need to re-create it, you might encounter this error:

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

This is a sign that your version of `yacc` is deficient. You probably need to install `bison` (the GNU version of `yacc`) and use that instead.

- On Debian Linux 3.0, you need to install `gawk` instead of the default `mawk`.
- If you need to debug `mysqld` or a MySQL client, run `configure` with the `--with-debug` option, and then recompile and link your clients with the new client library. See [MySQL Internals: Porting](#).

- If you get a compilation error on Linux (for example, SuSE Linux 8.1 or Red Hat Linux 7.3) similar to the following one, you probably do not have `g++` installed:

```
libmysql.c:1329: warning: passing arg 5 of `gethostbyname_r' from
incompatible pointer type
libmysql.c:1329: too few arguments to function `gethostbyname_r'
libmysql.c:1329: warning: assignment makes pointer from integer
without a cast
make[2]: *** [libmysql.lo] Error 1
```

By default, the `configure` script attempts to determine the correct number of arguments by using `g++` (the GNU C++ compiler). This test yields incorrect results if `g++` is not installed. There are two ways to work around this problem:

- Make sure that the GNU C++ `g++` is installed. On some Linux distributions, the required package is called `gpp`; on others, it is named `gcc-c++`.
- Use `gcc` as your C++ compiler by setting the `CXX` environment variable to `gcc`:

```
export CXX="gcc"
```

You must run `configure` again after making either of those changes.

2.9.5. MIT-pthreads Notes

This section describes some of the issues involved in using MIT-pthreads.

On Linux, you should *not* use MIT-pthreads. Use the installed LinuxThreads implementation instead. See [Section 2.13.1, “Linux Notes”](#).

If your system does not provide native thread support, you should build MySQL using the MIT-pthreads package. This includes older FreeBSD systems, SunOS 4.x, Solaris 2.4 and earlier, and some others. See [Section 2.1.1, “Operating Systems Supported by MySQL Community Server”](#).

MIT-pthreads is not part of the MySQL 5.1 source distribution. If you require this package, you need to download it separately from http://dev.mysql.com/Downloads/Contrib/pthreads-1_60_beta6-mysql.tar.gz

After downloading, extract this source archive into the top level of the MySQL source directory. It creates a new subdirectory named `mit-pthreads`.

- On most systems, you can force MIT-pthreads to be used by running `configure` with the `--with-mit-threads` option:

```
shell> ./configure --with-mit-threads
```

Building in a non-source directory is not supported when using MIT-pthreads because we want to minimize our changes to this code.

- The checks that determine whether to use MIT-pthreads occur only during the part of the configuration process that deals with the server code. If you have configured the distribution using `--without-server` to build only the client code, clients do not know whether MIT-pthreads is being used and use Unix socket file connections by default. Because Unix socket files do not work under MIT-pthreads on some platforms, this means you need to use `-h` or `--host` with a value other than `localhost` when you run client programs.
- When MySQL is compiled using MIT-pthreads, system locking is disabled by default for performance reasons. You can tell the server to use system locking with the `--external-locking` option. This is needed only if you want to be able to run two MySQL servers against the same data files, but that is not recommended, anyway.
- Sometimes the pthread `bind()` command fails to bind to a socket without any error message (at least on Solaris). The result is that all connections to the server fail. For example:

```
shell> mysqladmin version
mysqladmin: connect to server at '' failed;
error: 'Can't connect to mysql server on localhost (146)'
```

The solution to this problem is to kill the `mysqld` server and restart it. This has happened to us only when we have forcibly stopped the server and restarted it immediately.

- With MIT-pthreads, the `sleep()` system call isn't interruptible with `SIGINT` (break). This is noticeable only when you run `mysqladmin --sleep`. You must wait for the `sleep()` call to terminate before the interrupt is served and the process stops.
- When linking, you might receive warning messages like these (at least on Solaris); they can be ignored:

```
ld: warning: symbol `__iob' has differing sizes:
(file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
/my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
ld: warning: symbol `__iob' has differing sizes:
(file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
/my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
```

- Some other warnings also can be ignored:

```
implicit declaration of function `int strtoll(...)'
implicit declaration of function `int strtoul(...)'
```

- We have not been able to make `readline` work with MIT-pthreads. (This is not necessary, but may be of interest to some.)

2.9.6. Installing MySQL from Source on Windows

These instructions describe how to build binaries from source for MySQL 5.1 on Windows. Instructions are provided for building binaries from a standard source distribution or from the BitKeeper tree that contains the latest development source.

Note

The instructions here are strictly for users who want to test MySQL on Microsoft Windows from the latest source distribution or from the BitKeeper tree. For production use, MySQL AB does not advise using a MySQL server built by yourself from source. Normally, it is best to use precompiled binary distributions of MySQL that are built specifically for optimal performance on Windows by MySQL AB. Instructions for installing binary distributions are available in [Section 2.3](#), “Installing MySQL on Windows”.

To build MySQL on Windows from source, you must satisfy the following system, compiler, and resource requirements:

- Windows 2000, Windows XP, or newer version. Windows Vista is not supported until Microsoft certifies Visual Studio 2005 on Vista.
- CMake, which can be downloaded from <http://www.cmake.org>. After installing, modify your path to include the `cmake` binary.
- Microsoft Visual C++ 2005 Express Edition, Visual Studio .Net 2003 (7.1), or Visual Studio 2005 (8.0) compiler system.
- If you are using Visual C++ 2005 Express Edition, you must also install an appropriate Platform SDK. More information and links to downloads for various Windows platforms is available from <http://msdn.microsoft.com/platformsdk/>.
- If you are compiling from a BitKeeper tree or making changes to the parser, you need `bison` for Windows, which can be downloaded from <http://gnuwin32.sourceforge.net/packages/bison.htm>. Download the package labeled “Complete package, excluding sources”. After installing the package, modify your path to include the `bison` binary and ensure that this binary is accessible from Visual Studio.
- Cygwin might be necessary if you want to run the test script or package the compiled binaries and support files into a Zip archive. (Cygwin is needed only to test or package the distribution, not to build it.) Cygwin is available from <http://cygwin.com>.
- 3GB to 5GB of disk space.

The exact system requirements can be found here: <http://msdn.microsoft.com/vstudio/Previous/2003/sysreqs/default.aspx> and <http://msdn.microsoft.com/vstudio/products/sysreqs/default.aspx>

You also need a MySQL source distribution for Windows, which can be obtained two ways:

- Obtain a source distribution packaged by MySQL AB. These are available from <http://dev.mysql.com/downloads/>.
- Package a source distribution yourself from the latest BitKeeper developer source tree. For instructions on pulling the latest source files, see [Section 2.9.3, “Installing from the Development Source Tree”](#).

If you find something not working as expected, or you have suggestions about ways to improve the current build process on Windows, please send a message to the [win32](#) mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#).

2.9.6.1. Building MySQL from Source Using CMake and Visual Studio

You can build MySQL on Windows by using a combination of [cmake](#) and Microsoft Visual Studio .NET 2003 (7.1), Microsoft Visual Studio 2005 (8.0) or Microsoft Visual C++ 2005 Express Edition. You must have the appropriate Microsoft Platform SDK installed.

Note

To compile from the source code on Windows you must use the standard source distribution (for example, [mysql-5.0.45.tar.gz](#)). You build from the same distribution as used to build MySQL on Unix, Linux and other platforms. Do not use the Windows Source distributions as they do not contain the necessary configuration script and other files.

Follow this procedure to build MySQL:

1. If you are installing from a packaged source distribution, create a work directory (for example, `C:\workdir`), and unpack the source distribution there using [WinZip](#) or another Windows tool that can read `.zip` files. This directory is the work directory in the following instructions.
2. If you are installing from a BitKeeper tree, the root directory of that tree is the work directory in the following instructions.
3. Using a command shell, navigate to the work directory and run the following command:

```
C:\workdir>win\configure options
```

These options are available:

- `WITH_INNOBASE_STORAGE_ENGINE`: Enable the `InnoDB` storage engine.
- `WITH_PARTITION_STORAGE_ENGINE`: Enable user-defined partitioning.
- `WITH_ARCHIVE_STORAGE_ENGINE`: Enable the `ARCHIVE` storage engine.
- `WITH_BLACKHOLE_STORAGE_ENGINE`: Enable the `BLACKHOLE` storage engine.
- `WITH_EXAMPLE_STORAGE_ENGINE`: Enable the `EXAMPLE` storage engine.
- `WITH_FEDERATED_STORAGE_ENGINE`: Enable the `FEDERATED` storage engine.
- `__NT__`: Enable support for named pipes.
- `MYSQL_SERVER_SUFFIX=suffix`: Server suffix, default none.
- `COMPILATION_COMMENT=comment`: Server comment, default "Source distribution".
- `MYSQL_TCP_PORT=port`: Server port, default 3306.
- `DISABLE_GRANT_OPTIONS`: Disables the `--bootstrap`, `--skip-grant-tables`, and `--init-file` options for `mysqld`. This option is available as of MySQL 5.1.15.

For example (type the command on one line):

```
C:\workdir>win\configure WITH_INNOBASE_STORAGE_ENGINE  
WITH_PARTITION_STORAGE_ENGINE MYSQL_SERVER_SUFFIX=-pro
```

4. From the work directory, execute the `win\build-vs8.bat` or `win\build-vs71.bat` file, depending on the version of

Visual Studio you have installed. The script invokes CMake, which generates the `mysql.sln` solution file.

You can also use `win\build-vs8_x64.bat` to build the 64-bit version of MySQL. However, you cannot build the 64-bit version with Visual Studio Express Edition. You must use Visual Studio 2005 (8.0) or higher.

5. From the work directory, open the generated `mysql.sln` file with Visual Studio and select the proper configuration using the **CONFIGURATION** menu. The menu provides Debug, Release, RelwithDebInfo, MinRelInfo options. Then select **SOLUTION > Build** to build the solution.

Remember the configuration that you use in this step. It is important later when you run the test script because that script needs to know which configuration you used.

6. Test the server. The server built using the preceding instructions expects that the MySQL base directory and data directory are `C:\mysql` and `C:\mysql\data` by default. If you want to test your server using the source tree root directory and its data directory as the base directory and data directory, you need to tell the server their pathnames. You can either do this on the command line with the `--basedir` and `--datadir` options, or by placing appropriate options in an option file. (See [Section 4.3.2, “Using Option Files”](#).) If you have an existing data directory elsewhere that you want to use, you can specify its pathname instead.

When the server is running in standalone fashion or as a service based on your configuration, try to connect to it from the `mysql` interactive command-line utility.

You can also run the standard test script, `mysql-test-run.pl`. This script is written in Perl, so you'll need either Cygwin or ActiveState Perl to run it. You may also need to install the modules required by the script. To run the test script, change location into the `mysql-test` directory under the work directory, set the `MTR_VS_CONFIG` environment variable to the configuration you selected earlier (or use the `--vs-config` option), and invoke `mysql-test-run.pl`. For example (using Cygwin and the `bash` shell):

```
shell> cd mysql-test
shell> export MTR_VS_CONFIG=debug
shell> ./mysqltest-run.pl --force --timer
shell> ./mysqltest-run.pl --force --timer --ps-protocol
```

When you are satisfied that the programs you have built are working correctly, stop the server. Now you can install the distribution. One way to do this is to use the `make_win_bin_dist` script in the `scripts` directory of the MySQL source distribution (see [Section 5.5.2, “make_win_bin_dist — Package MySQL Distribution as ZIP Archive”](#)). This is a shell script, so you must have Cygwin installed if you want to use it. It creates a Zip archive of the built executables and support files that you can unpack in the location at which you want to install MySQL.

It is also possible to install MySQL by copying directories and files directly:

1. Create the directories where you want to install MySQL. For example, to install into `C:\mysql`, use these commands:

```
C:\> mkdir C:\mysql
C:\> mkdir C:\mysql\bin
C:\> mkdir C:\mysql\data
C:\> mkdir C:\mysql\share
C:\> mkdir C:\mysql\scripts
```

If you want to compile other clients and link them to MySQL, you should also create several additional directories:

```
C:\> mkdir C:\mysql\include
C:\> mkdir C:\mysql\lib
C:\> mkdir C:\mysql\lib\debug
C:\> mkdir C:\mysql\lib\opt
```

If you want to benchmark MySQL, create this directory:

```
C:\> mkdir C:\mysql\sql-bench
```

Benchmarking requires Perl support. See [Section 2.15, “Perl Installation Notes”](#).

2. From the work directory, copy into the `C:\mysql` directory the following directories:

```
C:\> cd \workdir
C:\workdir> copy client_release\*.exe C:\mysql\bin
C:\workdir> copy client_debug\mysqld.exe C:\mysql\bin\mysqld-debug.exe
```

```
C:\workdir> xcopy scripts\*. * C:\mysql\scripts /E
C:\workdir> xcopy share\*. * C:\mysql\share /E
```

If you want to compile other clients and link them to MySQL, you should also copy several libraries and header files:

```
C:\workdir> copy lib_debug\mysqlclient.lib C:\mysql\lib\debug
C:\workdir> copy lib_debug\libmysql.* C:\mysql\lib\debug
C:\workdir> copy lib_debug\zlib.* C:\mysql\lib\debug
C:\workdir> copy lib_release\mysqlclient.lib C:\mysql\lib\opt
C:\workdir> copy lib_release\libmysql.* C:\mysql\lib\opt
C:\workdir> copy lib_release\zlib.* C:\mysql\lib\opt
C:\workdir> copy include\*.h C:\mysql\include
C:\workdir> copy libmysql\libmysql.def C:\mysql\include
```

If you want to benchmark MySQL, you should also do this:

```
C:\workdir> xcopy sql-bench\*. * C:\mysql\bench /E
```

After installation, set up and start the server in the same way as for binary Windows distributions. See [Section 2.3, “Installing MySQL on Windows”](#).

2.9.7. Compiling MySQL Clients on Windows

In your source files, you should include `my_global.h` before `mysql.h`:

```
#include <my_global.h>
#include <mysql.h>
```

`my_global.h` includes any other files needed for Windows compatibility (such as `windows.h`) if you compile your program on Windows.

You can either link your code with the dynamic `libmysql.lib` library, which is just a wrapper to load in `libmysql.dll` on demand, or link with the static `mysqlclient.lib` library.

The MySQL client libraries are compiled as threaded libraries, so you should also compile your code to be multi-threaded.

2.10. Post-Installation Setup and Testing

After installing MySQL, there are some issues that you should address. For example, on Unix, you should initialize the data directory and create the MySQL grant tables. On all platforms, an important security concern is that the initial accounts in the grant tables have no passwords. You should assign passwords to prevent unauthorized access to the MySQL server. Optionally, you can create time zone tables to enable recognition of named time zones.

The following sections include post-installation procedures that are specific to Windows systems and to Unix systems. Another section, [Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”](#), applies to all platforms; it describes what to do if you have trouble getting the server to start. [Section 2.10.3, “Securing the Initial MySQL Accounts”](#), also applies to all platforms. You should follow its instructions to make sure that you have properly protected your MySQL accounts by assigning passwords to them.

When you are ready to create additional user accounts, you can find information on the MySQL access control system and account management in [Section 5.7, “The MySQL Access Privilege System”](#), and [Section 5.8, “MySQL User Account Management”](#).

2.10.1. Windows Post-Installation Procedures

On Windows, the data directory and the grant tables do not have to be created. MySQL Windows distributions include the grant tables with a set of preinitialized accounts in the `mysql` database under the data directory. It is unnecessary to run the `mysql_install_db` script that is used on Unix. Regarding passwords, if you installed MySQL using the Windows Installation Wizard, you may have already assigned passwords to the accounts. (See [Section 2.3.3, “Using the MySQL Installation Wizard”](#).) Otherwise, use the password-assignment procedure given in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

Before setting up passwords, you might want to try running some client programs to make sure that you can connect to the server and that it is operating properly. Make sure that the server is running (see [Section 2.3.9, “Starting the Server for the First Time”](#)), and then issue the following commands to verify that you can retrieve information from the server. The output should be similar to what is shown here:

```

C:\> C:\mysql\bin\mysqlshow
+-----+
| Databases |
+-----+
| mysql    |
| test     |
+-----+

C:\> C:\mysql\bin\mysqlshow mysql
Database: mysql
+-----+
| Tables |
+-----+
| columns_priv |
| db           |
| func         |
| help_category |
| help_keyword |
| help_relation |
| help_topic   |
| host         |
| proc         |
| procs_priv   |
| tables_priv  |
| time_zone    |
| time_zone_leap_second |
| time_zone_name |
| time_zone_transition |
| time_zone_transition_type |
| user         |
+-----+

C:\> C:\mysql\bin\mysql -e "SELECT Host,Db,User FROM db" mysql
+-----+-----+-----+
| host | db   | user |
+-----+-----+-----+
| %    | test% |      |
+-----+-----+-----+

```

If you are running a version of Windows that supports services and you want the MySQL server to run automatically when Windows starts, see [Section 2.3.11, “Starting MySQL as a Windows Service”](#).

2.10.2. Unix Post-Installation Procedures

After installing MySQL on Unix, you need to initialize the grant tables, start the server, and make sure that the server works satisfactorily. You may also wish to arrange for the server to be started and stopped automatically when your system starts and stops. You should also assign passwords to the accounts in the grant tables.

On Unix, the grant tables are set up by the `mysql_install_db` program. For some installation methods, this program is run for you automatically:

- If you install MySQL on Linux using RPM distributions, the server RPM runs `mysql_install_db`.
- If you install MySQL on Mac OS X using a PKG distribution, the installer runs `mysql_install_db`.

Otherwise, you'll need to run `mysql_install_db` yourself.

The following procedure describes how to initialize the grant tables (if that has not previously been done) and then start the server. It also suggests some commands that you can use to test whether the server is accessible and working properly. For information about starting and stopping the server automatically, see [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).

After you complete the procedure and have the server running, you should assign passwords to the accounts created by `mysql_install_db`. Instructions for doing so are given in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

In the examples shown here, the server runs under the user ID of the `mysql` login account. This assumes that such an account exists. Either create the account if it does not exist, or substitute the name of a different existing login account that you plan to use for running the server.

1. Change location into the top-level directory of your MySQL installation, represented here by *BASEDIR*:

```
shell> cd BASEDIR
```

`BASEDIR` is likely to be something like `/usr/local/mysql` or `/usr/local`. The following steps assume that you are located in this directory.

2. If necessary, run the `mysql_install_db` program to set up the initial MySQL grant tables containing the privileges that determine how users are allowed to connect to the server. You'll need to do this if you used a distribution type for which the installation procedure doesn't run the program for you.

Typically, `mysql_install_db` needs to be run only the first time you install MySQL, so you can skip this step if you are upgrading an existing installation. However, `mysql_install_db` does not overwrite any existing privilege tables, so it should be safe to run in any circumstances.

To initialize the grant tables, use one of the following commands, depending on whether `mysql_install_db` is located in the `bin` or `scripts` directory:

```
shell> bin/mysql_install_db --user=mysql
shell> scripts/mysql_install_db --user=mysql
```

The `mysql_install_db` script creates the server's data directory. Under the data directory, it creates directories for the `mysql` database that holds all database privileges and the `test` database that you can use to test MySQL. The script also creates privilege table entries for `root` and anonymous-user accounts. The accounts have no passwords initially. A description of their initial privileges is given in [Section 2.10.3, "Securing the Initial MySQL Accounts"](#). Briefly, these privileges allow the MySQL `root` user to do anything, and allow anybody to create or use databases with a name of `test` or starting with `test_`.

It is important to make sure that the database directories and files are owned by the `mysql` login account so that the server has read and write access to them when you run it later. To ensure this, the `--user` option should be used as shown if you run `mysql_install_db` as `root`. Otherwise, you should execute the script while logged in as `mysql`, in which case you can omit the `--user` option from the command.

`mysql_install_db` creates several tables in the `mysql` database, including `user`, `db`, `host`, `tables_priv`, `columns_priv`, `func`, and others. See [Section 5.7, "The MySQL Access Privilege System"](#), for a complete listing and description of these tables.

If you don't want to have the `test` database, you can remove it with `mysqladmin -u root drop test` after starting the server.

If you have trouble with `mysql_install_db` at this point, see [Section 2.10.2.1, "Problems Running mysql_install_db"](#).

3. Start the MySQL server:

```
shell> bin/mysqld_safe --user=mysql &
```

It is important that the MySQL server be run using an unprivileged (non-`root`) login account. To ensure this, the `--user` option should be used as shown if you run `mysqld_safe` as system `root`. Otherwise, you should execute the script while logged in to the system as `mysql`, in which case you can omit the `--user` option from the command.

Further instructions for running MySQL as an unprivileged user are given in [Section 5.6.5, "How to Run MySQL as a Normal User"](#).

If you neglected to create the grant tables before proceeding to this step, the following message appears in the error log file when you start the server:

```
mysqld: Can't find file: 'host.frm'
```

If you have other problems starting the server, see [Section 2.10.2.3, "Starting and Troubleshooting the MySQL Server"](#).

4. Use `mysqladmin` to verify that the server is running. The following commands provide simple tests to check whether the server is up and responding to connections:

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

The output from `mysqladmin version` varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

```

shell> bin/mysqladmin version
mysqladmin Ver 14.12 Distrib 5.1.21-beta, for pc-linux-gnu on i686
Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license

Server version          5.1.21-beta-Max
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 14 days 5 hours 5 min 21 sec

Threads: 1 Questions: 366 Slow queries: 0
Opens: 0 Flush tables: 1 Open tables: 19
Queries per second avg: 0.000

```

To see what else you can do with `mysqladmin`, invoke it with the `--help` option.

5. Verify that you can shut down the server:

```
shell> bin/mysqladmin -u root shutdown
```

6. Verify that you can start the server again. Do this by using `mysqld_safe` or by invoking `mysqld` directly. For example:

```
shell> bin/mysqld_safe --user=mysql --log &
```

If `mysqld_safe` fails, see [Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”](#).

7. Run some simple tests to verify that you can retrieve information from the server. The output should be similar to what is shown here:

```

shell> bin/mysqlshow
+-----+
| Databases |
+-----+
| mysql    |
| test     |
+-----+

shell> bin/mysqlshow mysql
Database: mysql
+-----+
| Tables |
+-----+
| columns_priv |
| db            |
| func          |
| help_category |
| help_keyword  |
| help_relation |
| help_topic    |
| host          |
| proc          |
| procs_priv    |
| tables_priv   |
| time_zone     |
| time_zone_leap_second |
| time_zone_name |
| time_zone_transition |
| time_zone_transition_type |
| user          |
+-----+

shell> bin/mysql -e "SELECT Host,Db,User FROM db" mysql
+-----+-----+-----+
| host | db   | user |
+-----+-----+-----+
| %    | test |      |
| %    | test_% |      |
+-----+-----+-----+

```

8. There is a benchmark suite in the `sql-bench` directory (under the MySQL installation directory) that you can use to compare how MySQL performs on different platforms. The benchmark suite is written in Perl. It requires the Perl DBI module that provides a database-independent interface to the various databases, and some other additional Perl modules:

```

DBI
DBD::mysql
Data::Dumper

```

```
Data::ShowTable
```

These modules can be obtained from CPAN (<http://www.cpan.org/>). See also [Section 2.15.1, “Installing Perl on Unix”](#).

The `sql-bench/Results` directory contains the results from many runs against different databases and platforms. To run all tests, execute these commands:

```
shell> cd sql-bench
shell> perl run-all-tests
```

If you don't have the `sql-bench` directory, you probably installed MySQL using RPM files other than the source RPM. (The source RPM includes the `sql-bench` benchmark directory.) In this case, you must first install the benchmark suite before you can use it. There are separate benchmark RPM files named `mysql-bench-VERSION.i386.rpm` that contain benchmark code and data.

If you have a source distribution, there are also tests in its `tests` subdirectory that you can run. For example, to run `auto_increment.tst`, execute this command from the top-level directory of your source distribution:

```
shell> mysql -vxf test < ./tests/auto_increment.tst
```

The expected result of the test can be found in the `./tests/auto_increment.res` file.

9. At this point, you should have the server running. However, none of the initial MySQL accounts have a password, so you should assign passwords using the instructions found in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

The MySQL 5.1 installation procedure creates time zone tables in the `mysql` database. However, you must populate the tables manually using the instructions in [Section 5.10.8, “MySQL Server Time Zone Support”](#).

2.10.2.1. Problems Running `mysql_install_db`

The purpose of the `mysql_install_db` script is to generate new MySQL privilege tables. It does not overwrite existing MySQL privilege tables, and it does not affect any other data.

If you want to re-create your privilege tables, first stop the `mysqld` server if it's running. Then rename the `mysql` directory under the data directory to save it, and then run `mysql_install_db`. Suppose that your current directory is the MySQL installation directory and that `mysql_install_db` is located in the `bin` directory and the data directory is named `data`. To rename the `mysql` database and re-run `mysql_install_db`, use these commands.

```
shell> mv data/mysql data/mysql.old
shell> bin/mysql_install_db --user=mysql
```

When you run `mysql_install_db`, you might encounter the following problems:

- **`mysql_install_db` fails to install the grant tables**

You may find that `mysql_install_db` fails to install the grant tables and terminates after displaying the following messages:

```
Starting mysqld daemon with databases from XXXXXX
mysqld ended
```

In this case, you should examine the error log file very carefully. The log should be located in the directory `XXXXXX` named by the error message and should indicate why `mysqld` didn't start. If you do not understand what happened, include the log when you post a bug report. See [Section 1.8, “How to Report Bugs or Problems”](#).

- **There is a `mysqld` process running**

This indicates that the server is running, in which case the grant tables have probably been created already. If so, there is no need to run `mysql_install_db` at all because it needs to be run only once (when you install MySQL the first time).

- **Installing a second `mysqld` server does not work when one server is running**

This can happen when you have an existing MySQL installation, but want to put a new installation in a different location. For example, you might have a production installation, but you want to create a second installation for testing purposes. Generally the problem that occurs when you try to run a second server is that it tries to use a network interface that is in use by the first server. In this case, you should see one of the following error messages:

```
Can't start server: Bind on TCP/IP port:
Address already in use
Can't start server: Bind on unix socket...
```

For instructions on setting up multiple servers, see [Section 5.12, “Running Multiple MySQL Servers on the Same Machine”](#).

- **You do not have write access to the `/tmp` directory**

If you do not have write access to create temporary files or a Unix socket file in the default location (the `/tmp` directory), an error occurs when you run `mysql_install_db` or the `mysqld` server.

You can specify different locations for the temporary directory and Unix socket file by executing these commands prior to starting `mysql_install_db` or `mysqld`, where *some_tmp_dir* is the full pathname to some directory for which you have write permission:

```
shell> TMPDIR=/some_tmp_dir/
shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock
shell> export TMPDIR MYSQL_UNIX_PORT
```

Then you should be able to run `mysql_install_db` and start the server with these commands:

```
shell> bin/mysql_install_db --user=mysql
shell> bin/mysqld_safe --user=mysql &
```

If `mysql_install_db` is located in the `scripts` directory, modify the first command to `scripts/mysql_install_db`.

See [Section B.1.4.5, “How to Protect or Change the MySQL Unix Socket File”](#), and [Section 2.14, “Environment Variables”](#).

There are some alternatives to running the `mysql_install_db` script provided in the MySQL distribution:

- If you want the initial privileges to be different from the standard defaults, you can modify `mysql_install_db` before you run it. However, it is preferable to use `GRANT` and `REVOKE` to change the privileges *after* the grant tables have been set up. In other words, you can run `mysql_install_db`, and then use `mysql -u root mysql` to connect to the server as the MySQL `root` user so that you can issue the necessary `GRANT` and `REVOKE` statements.

If you want to install MySQL on several machines with the same privileges, you can put the `GRANT` and `REVOKE` statements in a file and execute the file as a script using `mysql` after running `mysql_install_db`. For example:

```
shell> bin/mysql_install_db --user=mysql
shell> bin/mysql -u root < your_script_file
```

By doing this, you can avoid having to issue the statements manually on each machine.

- It is possible to re-create the grant tables completely after they have previously been created. You might want to do this if you're just learning how to use `GRANT` and `REVOKE` and have made so many modifications after running `mysql_install_db` that you want to wipe out the tables and start over.

To re-create the grant tables, remove all the `.frm`, `.MYI`, and `.MYD` files in the `mysql` database directory. Then run the `mysql_install_db` script again.

- You can start `mysqld` manually using the `--skip-grant-tables` option and add the privilege information yourself using `mysql`:

```
shell> bin/mysqld_safe --user=mysql --skip-grant-tables &
shell> bin/mysql mysql
```

From `mysql`, manually execute the SQL commands contained in `mysql_install_db`. Make sure that you run `mysqladmin flush-privileges` or `mysqladmin reload` afterward to tell the server to reload the grant tables.

Note that by not using `mysql_install_db`, you not only have to populate the grant tables manually, you also have to create them first.

2.10.2.2. Starting and Stopping MySQL Automatically

Generally, you start the `mysqld` server in one of these ways:

- By invoking `mysqld` directly. This works on any platform.
- By running the MySQL server as a Windows service. The service can be set to start the server automatically when Windows starts, or as a manual service that you start on request. For instructions, see [Section 2.3.11, “Starting MySQL as a Windows Service”](#).
- By invoking `mysqld_safe`, which tries to determine the proper options for `mysqld` and then runs it with those options. This script is used on Unix and Unix-like systems. See [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#).
- By invoking `mysql.server`. This script is used primarily at system startup and shutdown on systems that use System V-style run directories, where it usually is installed under the name `mysql`. The `mysql.server` script starts the server by invoking `mysqld_safe`. See [Section 5.3.2, “mysql.server — MySQL Server Startup Script”](#).
- On Mac OS X, you can install a separate MySQL Startup Item package to enable the automatic startup of MySQL on system startup. The Startup Item starts the server by invoking `mysql.server`. See [Section 2.5, “Installing MySQL on Mac OS X”](#), for details.

The `mysqld_safe` and `mysql.server` scripts and the Mac OS X Startup Item can be used to start the server manually, or automatically at system startup time. `mysql.server` and the Startup Item also can be used to stop the server.

To start or stop the server manually using the `mysql.server` script, invoke it with `start` or `stop` arguments:

```
shell> mysql.server start
shell> mysql.server stop
```

Before `mysql.server` starts the server, it changes location to the MySQL installation directory, and then invokes `mysqld_safe`. If you want the server to run as some specific user, add an appropriate `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file, as shown later in this section. (It is possible that you will need to edit `mysql.server` if you've installed a binary distribution of MySQL in a non-standard location. Modify it to `cd` into the proper directory before it runs `mysqld_safe`. If you do this, your modified version of `mysql.server` may be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.)

`mysql.server stop` stops the server by sending a signal to it. You can also stop the server manually by executing `mysqladmin shutdown`.

To start and stop MySQL automatically on your server, you need to add start and stop commands to the appropriate places in your `/etc/rc*` files.

If you use the Linux server RPM package (`MySQL-server-VERSION.rpm`), the `mysql.server` script is installed in the `/etc/init.d` directory with the name `mysql`. You need not install it manually. See [Section 2.4, “Installing MySQL from RPM Packages on Linux”](#), for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as `mysqld`.

If you install MySQL from a source distribution or using a binary distribution format that does not install `mysql.server` automatically, you can install it manually. The script can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree.

To install `mysql.server` manually, copy it to the `/etc/init.d` directory with the name `mysql`, and then make it executable. Do this by changing location into the appropriate directory where `mysql.server` is located and executing these commands:

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```

Older Red Hat systems use the `/etc/rc.d/init.d` directory rather than `/etc/init.d`. Adjust the preceding commands accord-

ingly. Alternatively, first create `/etc/init.d` as a symbolic link that points to `/etc/rc.d/init.d`:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

After installing the script, the commands needed to activate it to run at system startup depend on your operating system. On Linux, you can use `chkconfig`:

```
shell> chkconfig --add mysql
```

On some Linux systems, the following command also seems to be necessary to fully enable the `mysql` script:

```
shell> chkconfig --level 345 mysql on
```

On FreeBSD, startup scripts generally should go in `/usr/local/etc/rc.d/`. The `rc(8)` manual page states that scripts in this directory are executed only if their basename matches the `*.sh` shell filename pattern. Any other files or directories present within the directory are silently ignored. In other words, on FreeBSD, you should install the `mysql.server` script as `/usr/local/etc/rc.d/mysql.server.sh` to enable automatic startup.

As an alternative to the preceding setup, some operating systems also use `/etc/rc.local` or `/etc/init.d/boot.local` to start additional services on startup. To start up MySQL using this method, you could append a command like the one following to the appropriate startup file:

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

For other systems, consult your operating system documentation to see how to install startup scripts.

You can add options for `mysql.server` in a global `/etc/my.cnf` file. A typical `/etc/my.cnf` file might look like this:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

The `mysql.server` script understands the following options: `basedir`, `datadir`, and `pid-file`. If specified, they *must* be placed in an option file, not on the command line. `mysql.server` understands only `start` and `stop` as command-line arguments.

The following table shows which option groups the server and each startup script read from option files:

Script	Option Groups
<code>mysqld</code>	<code>[mysqld]</code> , <code>[server]</code> , <code>[mysqld-major_version]</code>
<code>mysqld_safe</code>	<code>[mysqld]</code> , <code>[server]</code> , <code>[mysqld_safe]</code>
<code>mysql.server</code>	<code>[mysqld]</code> , <code>[mysql.server]</code> , <code>[server]</code>

`[mysqld-major_version]` means that groups with names like `[mysqld-5.0]` and `[mysqld-5.1]` are read by servers having versions 5.0.x, 5.1.x, and so forth. This feature can be used to specify options that can be read only by servers within a given release series.

For backward compatibility, `mysql.server` also reads the `[mysql_server]` group and `mysqld_safe` also reads the `[safe_mysqld]` group. However, you should update your option files to use the `[mysql.server]` and `[mysqld_safe]` groups instead when using MySQL 5.1.

See [Section 4.3.2, “Using Option Files”](#).

2.10.2.3. Starting and Troubleshooting the MySQL Server

This section provides troubleshooting suggestions for problems starting the server on Unix. If you are using Windows, see [Section 2.3.13, “Troubleshooting a MySQL Installation Under Windows”](#).

If you have problems starting the server, here are some things to try:

- Check the error log to see why the server does not start.
- Specify any special options needed by the storage engines you are using.
- Make sure that the server knows where to find the data directory.
- Make sure that the server can access the data directory. The ownership and permissions of the data directory and its contents must be set such that the server can read and modify them.
- Verify that the network interfaces the server wants to use are available.

Some storage engines have options that control their behavior. You can create a `my.cnf` file and specify startup options for the engines that you plan to use. If you are going to use storage engines that support transactional tables (**InnoDB**, **NDB**), be sure that you have them configured the way you want before starting the server:

MySQL Enterprise

For expert advice on start-up options appropriate to your circumstances, subscribe to The MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- If you are using **InnoDB** tables, see [Section 13.5.3, “InnoDB Configuration”](#).
- If you are using MySQL Cluster, see [Section 16.4, “MySQL Cluster Configuration”](#).

Storage engines will use default option values if you specify none, but it is recommended that you review the available options and specify explicit values for those for which the defaults are not appropriate for your installation.

When the `mysqld` server starts, it changes location to the data directory. This is where it expects to find databases and where it expects to write log files. The server also writes the pid (process ID) file in the data directory.

The data directory location is hardwired in when the server is compiled. This is where the server looks for the data directory by default. If the data directory is located somewhere else on your system, the server will not work properly. You can determine what the default path settings are by invoking `mysqld` with the `--verbose` and `--help` options.

If the default locations don't match the MySQL installation layout on your system, you can override them by specifying options to `mysqld` or `mysqld_safe` on the command line or in an option file.

To specify the location of the data directory explicitly, use the `--datadir` option. However, normally you can tell `mysqld` the location of the base directory under which MySQL is installed and it looks for the data directory there. You can do this with the `--basedir` option.

To check the effect of specifying path options, invoke `mysqld` with those options followed by the `--verbose` and `--help` options. For example, if you change location into the directory where `mysqld` is installed and then run the following command, it shows the effect of starting the server with a base directory of `/usr/local`:

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

You can specify other options such as `--datadir` as well, but `--verbose` and `--help` must be the last options.

Once you determine the path settings you want, start the server without `--verbose` and `--help`.

If `mysqld` is currently running, you can find out what path settings it is using by executing this command:

```
shell> mysqladmin variables
```

Or:

```
shell> mysqladmin -h host_name variables
```

`host_name` is the name of the MySQL server host.

If you get **Errcode 13** (which means **Permission denied**) when starting `mysqld`, this means that the privileges of the data directory or its contents do not allow the server access. In this case, you change the permissions for the involved files and directories so

that the server has the right to use them. You can also start the server as `root`, but this raises security issues and should be avoided.

On Unix, change location into the data directory and check the ownership of the data directory and its contents to make sure the server has access. For example, if the data directory is `/usr/local/mysql/var`, use this command:

```
shell> ls -la /usr/local/mysql/var
```

If the data directory or its files or subdirectories are not owned by the login account that you use for running the server, change their ownership to that account. If the account is named `mysql`, use these commands:

```
shell> chown -R mysql /usr/local/mysql/var
shell> chgrp -R mysql /usr/local/mysql/var
```

If the server fails to start up correctly, check the error log. Log files are located in the data directory (typically `C:\Program Files\MySQL\MySQL Server 5.1\data` on Windows, `/usr/local/mysql/data` for a Unix binary distribution, and `/usr/local/var` for a Unix source distribution). Look in the data directory for files with names of the form `host_name.err` and `host_name.log`, where `host_name` is the name of your server host. Then examine the last few lines of these files. On Unix, you can use `tail` to display them:

```
shell> tail host_name.err
shell> tail host_name.log
```

The error log should contain information that indicates why the server couldn't start.

If either of the following errors occur, it means that some other program (perhaps another `mysqld` server) is using the TCP/IP port or Unix socket file that `mysqld` is trying to use:

```
Can't start server: Bind on TCP/IP port: Address already in use
Can't start server: Bind on unix socket...
```

Use `ps` to determine whether you have another `mysqld` server running. If so, shut down the server before starting `mysqld` again. (If another server is running, and you really want to run multiple servers, you can find information about how to do so in [Section 5.12, “Running Multiple MySQL Servers on the Same Machine”](#).)

If no other server is running, try to execute the command `telnet your_host_name tcp_ip_port_number`. (The default MySQL port number is 3306.) Then press Enter a couple of times. If you don't get an error message like `telnet: Unable to connect to remote host: Connection refused`, some other program is using the TCP/IP port that `mysqld` is trying to use. You'll need to track down what program this is and disable it, or else tell `mysqld` to listen to a different port with the `--port` option. In this case, you'll also need to specify the port number for client programs when connecting to the server via TCP/IP.

Another reason the port might be inaccessible is that you have a firewall running that blocks connections to it. If so, modify the firewall settings to allow access to the port.

If the server starts but you can't connect to it, you should make sure that you have an entry in `/etc/hosts` that looks like this:

```
127.0.0.1      localhost
```

This problem occurs only on systems that do not have a working thread library and for which MySQL must be configured to use MIT-pthreads.

If you cannot get `mysqld` to start, you can try to make a trace file to find the problem by using the `--debug` option. See [MySQL Internals: Porting](#).

2.10.3. Securing the Initial MySQL Accounts

Part of the MySQL installation process is to set up the `mysql` database that contains the grant tables:

- Windows distributions contain preinitialized grant tables that are installed automatically.
- On Unix, the grant tables are populated by the `mysql_install_db` program. Some installation methods run this program for you. Others require that you execute it manually. For details, see [Section 2.10.2, “Unix Post-Installation Procedures”](#).

The grant tables define the initial MySQL user accounts and their access privileges. These accounts are set up as follows:

- Accounts with the username `root` are created. These are superuser accounts that can do anything. The initial `root` account passwords are empty, so anyone can connect to the MySQL server as `root` — *without a password* — and be granted all privileges.
- On Windows, one `root` account is created; this account allows connecting from the local host only. The Windows installer will optionally create an account allowing for connections from any host only if the user selects the **ENABLE ROOT ACCESS FROM REMOTE MACHINES** option during installation.
- On Unix, both `root` accounts are for connections from the local host. Connections must be made from the local host by specifying a hostname of `localhost` for one of the accounts, or the actual hostname or IP number for the other.
- Two anonymous-user accounts are created, each with an empty username. The anonymous accounts have no password, so anyone can use them to connect to the MySQL server.
- On Windows, one anonymous account is for connections from the local host. It has no global privileges. (Before MySQL 5.1.16, it has all global privileges, just like the `root` accounts.) The other is for connections from any host and has all privileges for the `test` database and for other databases with names that start with `test`.
- On Unix, both anonymous accounts are for connections from the local host. Connections must be made from the local host by specifying a hostname of `localhost` for one of the accounts, or the actual hostname or IP number for the other. These accounts have all privileges for the `test` database and for other databases with names that start with `test_`.

As noted, none of the initial accounts have passwords. This means that your MySQL installation is unprotected until you do something about it:

- If you want to prevent clients from connecting as anonymous users without a password, you should either assign a password to each anonymous account or else remove the accounts.
- You should assign a password to each MySQL `root` account.

The following instructions describe how to set up passwords for the initial MySQL accounts, first for the anonymous accounts and then for the `root` accounts. Replace “*newpwd*” in the examples with the actual password that you want to use. The instructions also cover how to remove the anonymous accounts, should you prefer not to allow anonymous access at all.

You might want to defer setting the passwords until later, so that you don't need to specify them while you perform additional setup or testing. However, be sure to set them before using your installation for production purposes.

Anonymous Account Password Assignment

To assign passwords to the anonymous accounts, connect to the server as `root` and then use either `SET PASSWORD` or `UPDATE`. In either case, be sure to encrypt the password using the `PASSWORD()` function.

To use `SET PASSWORD` on Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR '@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR '@%' = PASSWORD('newpwd');
```

To use `SET PASSWORD` on Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR '@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR '@'host_name' = PASSWORD('newpwd');
```

In the second `SET PASSWORD` statement, replace `host_name` with the name of the server host. This is the name that is specified in the `Host` column of the non-`localhost` record for `root` in the `user` table. If you don't know what hostname this is, issue the following statement before using `SET PASSWORD`:

```
mysql> SELECT Host, User FROM mysql.user;
```

Look for the record that has `root` in the `User` column and something other than `localhost` in the `Host` column. Then use that `Host` value in the second `SET PASSWORD` statement.

Anonymous Account Removal

If you prefer to remove the anonymous accounts instead, do so as follows:

```
shell> mysql -u root
mysql> DROP USER '';
```

The `DROP` statement applies both to Windows and to Unix. On Windows, if you want to remove only the anonymous account that has the same privileges as `root`, do this instead:

```
shell> mysql -u root
mysql> DROP USER ''@'localhost';
```

That account allows anonymous access but has full privileges, so removing it improves security.

`root` Account Password Assignment

You can assign passwords to the `root` accounts in several ways. The following discussion demonstrates three methods:

- Use the `SET PASSWORD` statement
- Use the `mysqladmin` command-line client program
- Use the `UPDATE` statement

To assign passwords using `SET PASSWORD`, connect to the server as `root` and issue two `SET PASSWORD` statements. Be sure to encrypt the password using the `PASSWORD()` function.

For Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('newpwd');
```

For Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

In the second `SET PASSWORD` statement, replace `host_name` with the name of the server host. This is the same hostname that you used when you assigned the anonymous account passwords.

To assign passwords to the `root` accounts using `mysqladmin`, execute the following commands:

```
shell> mysqladmin -u root password "newpwd"
shell> mysqladmin -u root -h host_name password "newpwd"
```

These commands apply both to Windows and to Unix. In the second command, replace `host_name` with the name of the server host. The double quotes around the password are not always necessary, but you should use them if the password contains spaces or other characters that are special to your command interpreter.

You can also use `UPDATE` to modify the `user` table directly. The following `UPDATE` statement assigns a password to both `root` accounts at once:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
-> WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

The `UPDATE` statement applies both to Windows and to Unix.

After the passwords have been set, you must supply the appropriate password whenever you connect to the server. For example, if you want to use `mysqladmin` to shut down the server, you can do so using this command:

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

Note

If you forget your `root` password after setting it up, [Section B.1.4.1, “How to Reset the Root Password”](#), covers the procedure for resetting it.

To set up additional accounts, you can use the `GRANT` statement. For instructions, see [Section 5.8.2, “Adding New User Accounts to MySQL”](#).

2.11. Upgrading MySQL

As a general rule, we recommend that when you upgrade from one release series to another, you should go to the next series rather than skipping a series. For example, if you currently are running MySQL 4.0 and wish to upgrade to a newer series, upgrade to MySQL 4.1 rather than to 5.0 or 5.1.

The following items form a checklist of things that you should do whenever you perform an upgrade:

- Before upgrading from MySQL 5.0 to 5.1, read [Section 2.11.1, “Upgrading from MySQL 5.0 to 5.1”](#), as well as [Appendix C, *MySQL Change History*](#). These provide information about features that are new in MySQL 5.1 or differ from those found in MySQL 5.0. If you wish to upgrade from a release series previous to MySQL 5.0, you should upgrade to each successive release series in turn until you have reached MySQL 5.0, and then proceed with the upgrade to MySQL 5.1. For information on upgrading from MySQL 5.0, see the *MySQL 5.0 Reference Manual*; for earlier releases, see the *MySQL 3.23, 4.0, 4.1 Reference Manual*.
- Before you perform an upgrade, back up your databases, including the `mysql` database that contains the grant tables.
- Some releases of MySQL introduce incompatible changes to tables. (Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.) Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features.

To avoid problems due to such changes, after you upgrade to a new version of MySQL, you should check your tables (and repair them if necessary), and update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 5.5.7, “`mysql\_upgrade` — Check Tables for MySQL Upgrade”](#).

- If you are running MySQL Server on Windows, see [Section 2.3.14, “Upgrading MySQL on Windows”](#).
- If you are using replication, see [Section 15.3.3, “Upgrading a Replication Setup”](#), for information on upgrading your replication setup.
- As of MySQL 5.1.9, the `mysqld-max` server is included in binary distributions. There is no separate MySQL-Max distribution. As of MySQL 5.1.12, binary distributions contain a server that includes the features previously included in `mysqld-max`.
- If you have created a user-defined function (UDF) with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF with a different non-conflicting name. The same is true if the new version of MySQL implements a built-in function with the same name as an existing stored function. See [Section 8.2.4, “Function Name Parsing and Resolution”](#), for the rules describing how the server interprets references to different kinds of functions.

You can always move the MySQL format files and data files between different versions on the same architecture as long as you stay within versions for the same release series of MySQL. If you change the character set when running MySQL, you must run `myisamchk -r -q --set-collation=collation_name` on all `MyISAM` tables. Otherwise, your indexes may not be ordered correctly, because changing the character set may also change the sort order.

If you are cautious about using new versions, you can always rename your old `mysqld` before installing a newer one. For example, if you are using MySQL 5.0.13 and want to upgrade to 5.1.10, rename your current server from `mysqld` to `mysqld-5.0.13`. If your new `mysqld` then does something unexpected, you can simply shut it down and restart with your old `mysqld`.

If, after an upgrade, you experience problems with recompiled client programs, such as `Commands out of sync` or unexpected core dumps, you probably have used old header or library files when compiling your programs. In this case, you should check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries.

If problems occur, such as that the new `mysqld` server does not start or that you cannot connect without a password, verify that you do not have an old `my.cnf` file from your previous installation. You can check this with the `--print-defaults` option (for example,

`mysqld --print-defaults`). If this command displays anything other than the program name, you have an active `my.cnf` file that affects server or client operation.

It is a good idea to rebuild and reinstall the Perl `DBD: :mysql` module whenever you install a new release of MySQL. The same applies to other MySQL interfaces as well, such as the PHP `mysql` extension and the Python `MySQLdb` module.

2.11.1. Upgrading from MySQL 5.0 to 5.1

When upgrading a 5.0 installation to 5.0.10 or above note that it is *necessary* to upgrade your grant tables. Otherwise, creating stored procedures and functions might not work. The procedure for doing this is described in [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

Note

It is good practice to back up your data before installing any new version of software. Although MySQL works very hard to ensure a high level of quality, you should protect your data by making a backup. MySQL recommends that you dump and reload your tables from any previous version to upgrade to 5.1.

In general, you should do the following when upgrading from MySQL 5.0 from 5.1:

- Check the items in [Section 2.11, “Upgrading MySQL”](#), to see whether any of them might affect your applications.
- Check the items in the change lists found later in this section to see whether any of them might affect your applications. Note particularly any that are marked **Incompatible change**. These result in incompatibilities with earlier versions of MySQL, and may require your attention *before you upgrade*.
- Some releases of MySQL introduce incompatible changes to tables. (Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.) Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features.

To avoid problems due to such changes, after you upgrade to a new version of MySQL, you should run `mysql_upgrade` to check your tables (and repair them if necessary), and to update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

- Read the MySQL 5.1 change history to see what significant new features you can use in 5.1. See [Section C.1, “Changes in release 5.1.x \(Development\)”](#).
- If you are running MySQL Server on Windows, see [Section 2.3.14, “Upgrading MySQL on Windows”](#).
- If you are using replication, see [Section 15.3.3, “Upgrading a Replication Setup”](#), for information on upgrading your replication setup.

The following lists describe changes that may affect applications and that you should watch out for when upgrading to MySQL 5.1.

Configuration Changes:

- Before MySQL 5.1.11, to build MySQL from source with SSL support enabled, you would invoke `configure` with either the `--with-openssl` or `--with-yassl` option. In MySQL 5.1.11, those options both have been replaced by the `--with-ssl` option. By default, `--with-ssl` causes the bundled yaSSL library to be used. To select OpenSSL instead, give the option as `--with-ssl=path`, where `path` is the directory where the OpenSSL header files and libraries are located.

Server Changes:

- **Incompatible change:** MySQL 5.1 implements support for a plugin API that allows the loading and unloading of components at runtime, without restarting the server. [Section 28.2, “The MySQL Plugin Interface”](#). The plugin API requires the `mysql.plugin` table. When upgrading from an older version of MySQL, you should run the `mysql_upgrade` command to create this table. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

Plugins are installed in the directory named by the `plugin_dir` system variable. This variable also controls the location from which the server loads user-defined functions (UDFs), which is a change from earlier versions of MySQL. That is, all UDF library files now must be installed in the plugin directory. When upgrading from an older version of MySQL, you must migrate your UDF

files to the plugin directory.

- **Incompatible change:** The `table_cache` system variable has been renamed to `table_open_cache`. Any scripts that refer to `table_cache` should be updated to use the new name.
- **Incompatible change:** Several issues were identified for stored programs (stored functions and procedures, triggers, and events) and views containing non-ASCII symbols. These issues involved conversion errors due to incomplete character set information when translating these objects to and from stored format. To address these problems, the representation for these objects was changed in MySQL 5.1.21. To avoid warnings from the server about the use of old definitions from any release prior to 5.1.21, you should dump stored programs and views with `mysqldump` after upgrading to 5.1.21 or higher, and then reload them to recreate them with new definitions. Invoke `mysqldump` with a `--default-character-set` option that names the non-ASCII character set that was used for the definitions when the objects were originally defined.
- **Incompatible change:** As of MySQL 5.1.20, `mysqld_safe` supports error logging to `syslog` on systems that support the `logger` command. The new `--syslog` and `--skip-syslog` options can be used instead of the `--log-error` option to control logging behavior, as described in [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#).

In 5.1.21 and up, the default is `--skip-syslog`, which is compatible with the default behavior of writing an error log file for releases prior to 5.1.20.

In 5.1.20 only, the following conditions apply: 1) The default is to use `syslog`, which is not compatible with releases prior to 5.1.20. 2) Logging to `syslog` may fail to operate correctly in some cases, so we recommend that you use `--skip-syslog` or `--log-error`. To maintain the older behavior if you were using no error-logging option, use `--skip-syslog`. If you were using `--log-error`, continue to use it.

- **Incompatible change:** As of MySQL 5.1.15, the following conditions apply to enabling the `read_only` system variable:
 - If you attempt to enable `read_only` while you have any explicit locks (acquired with `LOCK TABLES` or have a pending transaction, an error will occur.
 - If other clients hold explicit table locks or have pending transactions, the attempt to enable `read_only` blocks until the locks are released and the transactions end. While the attempt to enable `read_only` is pending, requests by other clients for table locks or to begin transactions also block until `read_only` has been set.
 - `read_only` can be enabled while you hold a global read lock (acquired with `FLUSH TABLES WITH READ LOCK`) because that does not involve table locks.

Previously, the attempt to enable `read_only` would return immediately even if explicit locks or transactions were pending, so some data changes could occur for statements executing in the server at the same time.

- **Incompatible change:** The number of function names affected by `IGNORE_SPACE` was reduced significantly in MySQL 5.1.13, from about 200 to about 30. (For details about `IGNORE_SPACE`, see [Section 8.2.4, “Function Name Parsing and Resolution”](#).) This change improves the consistency of parser operation. However, it also introduces the possibility of incompatibility for old SQL code that relies on the following conditions:
 - `IGNORE_SPACE` is disabled.
 - The presence or absence of whitespace following a function name is used to distinguish between a built-in function and stored function that have the same name (for example, `PI()` versus `PI ()`).

For functions that are no longer affected by `IGNORE_SPACE` as of MySQL 5.1.13, that strategy no longer works. Either of the following approaches can be used if you have code that is subject to the preceding incompatibility:

- If a stored function has a name that conflicts with a built-in function, refer to the stored function with a schema name qualifier, regardless of whether whitespace is present. For example, write `schema_name.PI()` or `schema_name.PI ()`.
- Alternatively, rename the stored function to use a non-conflicting name and change invocations of the function to use the new name.
- **Incompatible change:** For `utf8` columns, the full-text parser incorrectly considered several non-word punctuation and whitespace characters as word characters, causing some searches to return incorrect results. The fix involves a change to the full-text parser in MySQL 5.1.12, so as of 5.1.12, any tables that have `FULLTEXT` indexes on `utf8` columns must be repaired with `REPAIR TABLE`:

```
REPAIR TABLE tbl_name QUICK;
```


- **Incompatible change:** Storage engines can be pluggable at runtime, so the distinction between disabled and invalid storage engines no longer applies. As of MySQL 5.1.12, this affects the `NO_ENGINE_SUBSTITUTION` SQL mode, as described in [Section 5.2.6, “SQL Modes”](#).
- **Incompatible change:** The structure of `FULLTEXT` indexes has been changed in MySQL 5.1.6. After upgrading to MySQL 5.1.6 or greater, call the `REPAIR TABLE` statement for each table that contains any `FULLTEXT` indexes.
- **Incompatible change:** In MySQL 5.1.6, when log tables were implemented, the default log destination for the general query and slow query log was `TABLE`. As of MySQL 5.1.21, this default has been changed to `FILE`, which is compatible with MySQL 5.0, but incompatible with earlier releases of MySQL 5.1 from 5.1.6 to 5.1.20. If you are upgrading from MySQL 5.0 to this release, no logging option changes should be necessary. However, if you are upgrading from 5.1.6 through 5.1.20 to this release and were using `TABLE` logging, use the `--log-output=TABLE` option explicitly to preserve your server's table-logging behavior.
- **Incompatible change:** For `ENUM` columns that had enumeration values containing commas, the commas were mapped to 0xff internally. However, this rendered the commas indistinguishable from true 0xff characters in the values. This no longer occurs. However, the fix requires that you dump and reload any tables that have `ENUM` columns containing true 0xff in their values: Dump the tables using `mysqldump` with the current server before upgrading from a version of MySQL 5.1 older than 5.1.15 to version 5.1.15 or newer.
- As of MySQL 5.1.12, the `lc_time_names` system variable specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()` and `MONTHNAME()` functions. See [Section 5.10.9, “MySQL Server Locale Support”](#).
- As of MySQL 5.1.6, special characters in database and table identifiers are encoded when creating the corresponding directory names and filenames. This relaxes the restrictions on the characters that can appear in identifiers. See [Section 8.2.3, “Mapping of Identifiers to Filenames”](#). When you run `mysql_upgrade`, it will cause database and table names to be updated to the new format should they contain special characters.
- As of MySQL 5.1.9, `mysqld_safe` no longer implicitly invokes `mysqld-max` if it exists. Instead, it invokes `mysqld` unless a `-mysqld` or `--mysqld-version` option is given to specify another server explicitly. If you previously relied on the implicit invocation of `mysqld-max`, you should use an appropriate option now.

SQL Changes:

- **Important note:** Prior to MySQL 5.1.17, the parser accepted invalid code in SQL condition handlers, leading to server crashes or unexpected execution behavior in stored programs. Specifically, the parser allowed a condition handler to refer to labels for blocks that enclose the handler declaration. This was incorrect because block label scope does not include the code for handlers declared within the labeled block.

As of 5.1.17, the parser rejects this invalid construct, but if you upgrade in place (without dumping and reloading your databases), existing handlers that contain the construct still are invalid *even if they appear to function as you expect* and should be rewritten.

To find affected handlers, use `mysqldump` to dump all stored functions and procedures, triggers, and events. Then attempt to reload them into an upgraded server. Handlers that contain illegal label references will be rejected.

For more information about condition handlers and writing them to avoid invalid jumps, see [Section 19.2.8.2, “DECLARE Handlers”](#).

- **Incompatible change:** As of MySQL 5.1.8, `TYPE = engine_name` is still accepted as a synonym for the `ENGINE = engine_name` table option but generates a warning. You should note that this option is not available in MySQL 5.1.7, and **is to be removed altogether in MySQL 5.2, where it will produce a syntax error**.

`TYPE` has been deprecated since MySQL 4.0.

- **Incompatible change:** The namespace for triggers has changed in MySQL 5.0.10. Previously, trigger names had to be unique per table. Now they must be unique within the schema (database). An implication of this change is that `DROP TRIGGER` syntax now uses a schema name instead of a table name (schema name is optional and, if omitted, the current schema will be used).

When upgrading from a previous version of MySQL 5 to MySQL 5.0.10 or newer, you must drop all triggers and re-create them or `DROP TRIGGER` will not work after the upgrade. Here is a suggested procedure for doing this:

1. Upgrade to MySQL 5.0.10 or later to be able to access trigger information in the `INFORMATION_SCHEMA.TRIGGERS` table. (It should work even for pre-5.0.10 triggers.)

2. Dump all trigger definitions using the following **SELECT** statement:

```
SELECT CONCAT('CREATE TRIGGER ', t.TRIGGER_SCHEMA, ' ', t.TRIGGER_NAME,
              ' ', t.ACTION_TIMING, ' ', t.EVENT_MANIPULATION, ' ON ',
              t.EVENT_OBJECT_SCHEMA, ' ', t.EVENT_OBJECT_TABLE,
              ' FOR EACH ROW ', t.ACTION_STATEMENT, '///')
INTO OUTFILE '/tmp/triggers.sql'
FROM INFORMATION_SCHEMA.TRIGGERS AS t;
```

The statement uses **INTO OUTFILE**, so you must have the **FILE** privilege. The file will be created on the server host; use a different filename if you like. To be 100% safe, inspect the trigger definitions in the **triggers.sql** file, and perhaps make a backup of the file.

3. Stop the server and drop all triggers by removing all **.TRG** files in your database directories. Change location to your data directory and issue this command:

```
shell> rm */*.TRG
```

4. Start the server and re-create all triggers using the **triggers.sql** file: For example in my case it was:

```
mysql> delimiter // ;
mysql> source /tmp/triggers.sql //
```

5. Check that all triggers were successfully created using the **SHOW TRIGGERS** statement.

- **Incompatible change:** MySQL 5.1.6 introduces the **TRIGGER** privilege. Previously, the **SUPER** privilege was needed to create or drop triggers. Now those operations require the **TRIGGER** privilege. This is a security improvement because you no longer need to grant users the **SUPER** privilege to enable them to create triggers. However, the requirement that the account named in a trigger's **DEFINER** clause must have the **SUPER** privilege has changed to a requirement for the **TRIGGER** privilege. When upgrading from a previous version of MySQL 5.0 or 5.1 to MySQL 5.1.6 or newer, be sure to update your grant tables as described in [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#). This process assigns the **TRIGGER** privilege to all accounts that had the **SUPER** privilege. If you fail to update the grant tables, triggers may fail when activated. (After updating the grant tables, you can revoke the **SUPER** privilege from those accounts that no longer otherwise require it.)
- Some keywords are reserved in MySQL 5.1 that were not reserved in MySQL 5.0. See [Section 8.3, “Reserved Words”](#).
- The **LOAD DATA FROM MASTER** and **LOAD TABLE FROM MASTER** statements are deprecated. See [Section 12.6.2.2, “LOAD DATA FROM MASTER Syntax”](#), for recommended alternatives.
- The **INSTALL PLUGIN** and **UNINSTALL PLUGIN** statements that are used for the plugin API are new. So is the **WITH PARSER** clause for **FULLTEXT** index creation that associates a parser plugin with a full-text index. [Section 28.2, “The MySQL Plugin Interface”](#).

C API Changes:

- **Incompatible change:** As of MySQL 5.1.7, the `mysql_stmt_attr_get()` C API function returns a boolean rather than an unsigned int for **STMT_ATTR_UPDATE_MAX_LENGTH**. ([Bug#16144](#))

2.11.2. Copying MySQL Databases to Another Machine

You can copy the **.frm**, **.MYI**, and **.MYD** files for **MyISAM** tables between different architectures that support the same floating-point format. (MySQL takes care of any byte-swapping issues.) See [Section 13.4, “The MyISAM Storage Engine”](#).

In cases where you need to transfer databases between different architectures, you can use **mysqldump** to create a file containing SQL statements. You can then transfer the file to the other machine and feed it as input to the **mysql** client.

Use **mysqldump --help** to see what options are available. If you are moving the data to a newer version of MySQL, you should use **mysqldump --opt** to take advantage of any optimizations that result in a dump file that is smaller and can be processed more quickly.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
shell> mysqladmin -h 'other_hostname' create db_name
shell> mysqldump --opt db_name | mysql -h 'other_hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use these commands:

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other_hostname' --opt --compress db_name | mysql db_name
```

You can also store the dump in a file, transfer the file to the target machine, and then load the file into the database there. For example, you can dump a database to a compressed file on the source machine like this:

```
shell> mysqldump --quick db_name | gzip > db_name.gz
```

Transfer the file containing the database contents to the target machine and run these commands there:

```
shell> mysqladmin create db_name
shell> gunzip < db_name.gz | mysql db_name
```

You can also use `mysqldump` and `mysqlimport` to transfer the database. For large tables, this is much faster than simply using `mysqldump`. In the following commands, `DUMPDIR` represents the full pathname of the directory you use to store the output from `mysqldump`.

First, create the directory for the output files and dump the database:

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR db_name
```

Then transfer the files in the `DUMPDIR` directory to some corresponding directory on the target machine and load the files into MySQL there:

```
shell> mysqladmin create db_name           # create database
shell> cat DUMPDIR/*.sql | mysql db_name   # create tables in database
shell> mysqlimport db_name DUMPDIR/*.txt   # load data into tables
```

Do not forget to copy the `mysql` database because that is where the grant tables are stored. You might have to run commands as the MySQL `root` user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute `mysqladmin flush-privileges` so that the server reloads the grant table information.

2.12. Downgrading MySQL

This section describes what you should do to downgrade to an older MySQL version in the unlikely case that the previous version worked better than the new one.

If you are downgrading within the same release series (for example, from 5.0.13 to 5.0.12) the general rule is that you just have to install the new binaries on top of the old ones. There is no need to do anything with the databases. As always, however, it is always a good idea to make a backup.

The following items form a checklist of things you should do whenever you perform a downgrade:

- Read the upgrading section for the release series from which you are downgrading to be sure that it does not have any features you really need. [Section 2.11, “Upgrading MySQL”](#).
- If there is a downgrading section for that version, you should read that as well.

In most cases, you can move the MySQL format files and data files between different versions on the same architecture as long as you stay within versions for the same release series of MySQL.

If you downgrade from one release series to another, there may be incompatibilities in table storage formats. In this case, you can use `mysqldump` to dump your tables before downgrading. After downgrading, reload the dump file using `mysql` or `mysqlimport` to re-create your tables. For examples, see [Section 2.11.2, “Copying MySQL Databases to Another Machine”](#).

The normal symptom of a downward-incompatible table format change when you downgrade is that you can't open tables. In that case, use the following procedure:

1. Stop the older MySQL server that you are downgrading to.
2. Restart the newer MySQL server you are downgrading from.
3. Dump any tables that were inaccessible to the older server by using `mysqldump` to create a dump file.
4. Stop the newer MySQL server and restart the older one.
5. Reload the dump file into the older server. Your tables should be accessible.

2.12.1. Downgrading to MySQL 5.0

When downgrading to MySQL 5.0 from MySQL 5.1 or a later version, you should keep in mind the following issues relating to features found in MySQL 5.1 and later, but not in MySQL 5.0:

- **Event Scheduler.** MySQL 5.0 does not support scheduled events. If your databases contain scheduled event definitions, you should prevent them from being dumped when you use `mysqldump` by using the `--skip-events` option. (See [Section 7.13, “mysqldump — A Database Backup Program”](#).)
- **Partitioning.** MySQL 5.0 does not support user-defined partitioning. If a table was created as a partitioned table in 5.1 (or if a table created in a previous version of MySQL was altered to include partitions after an upgrade to 5.1), the table is accessible after downgrade only if you do one of the following:
 - Export the table using `mysqldump` and then drop it in MySQL 5.1; import the table again following the downgrade to MySQL 5.0.
 - Prior to the downgrade, remove the table's partitioning using `ALTER TABLE table_name REMOVE PARTITIONING`.
- **Stored routines.** MySQL 5.1.21 added a number of new columns to the `mysql.proc` table in which stored routine definitions are stored. If you are downgrading from MySQL 5.1.21 or later to MySQL 5.0, you cannot import the MySQL 5.1 routine definitions into MySQL 5.0.46 or earlier using the dump of `mysql.proc` created by `mysqldump` (such as when using the `--all-databases` option). Instead, you should run `mysqldump --routines` prior to performing the downgrade and run the stored routines DDL statements following the downgrade.

See [Bug#11986](#), [Bug#30029](#), and [Bug#30660](#), for more information.

2.13. Operating System-Specific Notes

2.13.1. Linux Notes

This section discusses issues that have been found to occur on Linux. The first few subsections describe general operating system-related issues, problems that can occur when using binary or source distributions, and post-installation issues. The remaining subsections discuss problems that occur with Linux on specific platforms.

Note that most of these problems occur on older versions of Linux. If you are running a recent version, you may see none of them.

2.13.1.1. Linux Operating System Notes

MySQL needs at least Linux version 2.0.

Warning

We have seen some strange problems with Linux 2.2.14 and MySQL on SMP systems. We also have reports from some MySQL users that they have encountered serious stability problems using MySQL with kernel 2.2.14. If you are using this kernel, you should upgrade to 2.2.19 (or newer) or to a 2.4 kernel. If you have a multiple-CPU box, you should seriously consider using 2.4 because it gives you a significant speed boost. Your system should be more stable.

When using LinuxThreads, you should see a minimum of three `mysqld` processes running. These are in fact threads. There is one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

2.13.1.2. Linux Binary Distribution Notes

The Linux-Intel binary and RPM releases of MySQL are configured for the highest possible speed. We are always trying to use the fastest stable compiler available.

The binary release is linked with `-static`, which means you do not normally need to worry about which version of the system libraries you have. You need not install LinuxThreads, either. A program linked with `-static` is slightly larger than a dynamically linked program, but also slightly faster (3-5%). However, one problem with a statically linked program is that you can't use user-defined functions (UDFs). If you are going to write or use UDFs (this is something for C or C++ programmers only), you must compile MySQL yourself using dynamic linking.

A known issue with binary distributions is that on older Linux systems that use `libc` (such as Red Hat 4.x or Slackware), you get some (non-fatal) issues with hostname resolution. If your system uses `libc` rather than `glibc2`, you probably will encounter some difficulties with hostname resolution and `getpwnam()`. This happens because `glibc` (unfortunately) depends on some external libraries to implement hostname resolution and `getpwent()`, even when compiled with `-static`. These problems manifest themselves in two ways:

- You may see the following error message when you run `mysql_install_db`:

```
Sorry, the host 'xxxx' could not be looked up
```

You can deal with this by executing `mysql_install_db --force`, which does not execute the `resolveip` test in `mysql_install_db`. The downside is that you cannot use hostnames in the grant tables: except for `localhost`, you must use IP numbers instead. If you are using an old version of MySQL that does not support `--force`, you must manually remove the `resolveip` test in `mysql_install` using a text editor.

- You also may see the following error when you try to run `mysqld` with the `--user` option:

```
getpwnam: No such file or directory
```

To work around this problem, start `mysqld` by using the `su` command rather than by specifying the `--user` option. This causes the system itself to change the user ID of the `mysqld` process so that `mysqld` need not do so.

Another solution, which solves both problems, is not to use a binary distribution. Obtain a MySQL source distribution (in RPM or `tar.gz` format) and install that instead.

On some Linux 2.2 versions, you may get the error `Resource temporarily unavailable` when clients make a great many new connections to a `mysqld` server over TCP/IP. The problem is that Linux has a delay between the time that you close a TCP/IP socket and the time that the system actually frees it. There is room for only a finite number of TCP/IP slots, so you encounter the resource-unavailable error if clients attempt too many new TCP/IP connections over a short period of time. For example, you may see the error when you run the MySQL `test-connect` benchmark over TCP/IP.

We have inquired about this problem a few times on different Linux mailing lists but have never been able to find a suitable resolution. The only known “fix” is for clients to use persistent connections, or, if you are running the database server and clients on the same machine, to use Unix socket file connections rather than TCP/IP connections.

2.13.1.3. Linux Source Distribution Notes

The following notes regarding `glibc` apply only to the situation when you build MySQL yourself. If you are running Linux on an x86 machine, in most cases it is much better for you to use our binary. We link our binaries against the best patched version of `glibc` we can find and with the best compiler options, in an attempt to make it suitable for a high-load server. For a typical user, even for setups with a lot of concurrent connections or tables exceeding the 2GB limit, our binary is the best choice in most cases. After reading the following text, if you are in doubt about what to do, try our binary first to determine whether it meets your needs. If you discover that it is not good enough, you may want to try your own build. In that case, we would appreciate a note about it so that we can build a better binary next time.

MySQL uses LinuxThreads on Linux. If you are using an old Linux version that doesn't have `glibc2`, you must install LinuxThreads before trying to compile MySQL. You can obtain LinuxThreads from <http://dev.mysql.com/downloads/os-linux.html>.

Note that `glibc` versions before and including version 2.1.1 have a fatal bug in `pthread_mutex_timedwait()` handling, which is used when `INSERT DELAYED` statements are issued. We recommend that you not use `INSERT DELAYED` before upgrading `glibc`.

Note that Linux kernel and the LinuxThread library can by default handle a maximum of 1,024 threads. If you plan to have more than 1,000 concurrent connections, you need to make some changes to LinuxThreads, as follows:

- Increase `PTHREAD_THREADS_MAX` in `sysdeps/unix/sysv/linux/bits/local_lim.h` to 4096 and decrease `STACK_SIZE` in `linuxthreads/internals.h` to 256KB. The paths are relative to the root of `glibc`. (Note that MySQL is not stable with 600-1000 connections if `STACK_SIZE` is the default of 2MB.)
- Recompile LinuxThreads to produce a new `libpthread.a` library, and relink MySQL against it.

There is another issue that greatly hurts MySQL performance, especially on SMP systems. The mutex implementation in LinuxThreads in `glibc` 2.1 is very poor for programs with many threads that hold the mutex only for a short time. This produces a paradoxical result: If you link MySQL against an unmodified LinuxThreads, removing processors from an SMP actually improves MySQL performance in many cases. We have made a patch available for `glibc` 2.1.3 to correct this behavior (<http://dev.mysql.com/Downloads/Linux/linuxthreads-2.1-patch>).

With `glibc` 2.2.2, MySQL uses the adaptive mutex, which is much better than even the patched one in `glibc` 2.1.3. Be warned, however, that under some conditions, the current mutex code in `glibc` 2.2.2 overspins, which hurts MySQL performance. The likelihood that this condition occurs can be reduced by re-nicing the `mysqld` process to the highest priority. We have also been able to correct the overspin behavior with a patch, available at <http://dev.mysql.com/Downloads/Linux/linuxthreads-2.2.2.patch>. It combines the correction of overspin, maximum number of threads, and stack spacing all in one. You need to apply it in the `linuxthreads` directory with `patch -p0 </tmp/linuxthreads-2.2.2.patch`. We hope it is included in some form in future releases of `glibc` 2.2. In any case, if you link against `glibc` 2.2.2, you still need to correct `STACK_SIZE` and `PTHREAD_THREADS_MAX`. We hope that the defaults is corrected to some more acceptable values for high-load MySQL setup in the future, so that the commands needed to produce your own build can be reduced to `./configure; make; make install`.

We recommend that you use these patches to build a special static version of `libpthread.a` and use it only for statically linking against MySQL. We know that these patches are safe for MySQL and significantly improve its performance, but we cannot say anything about their effects on other applications. If you link other applications that require LinuxThreads against the patched static version of the library, or build a patched shared version and install it on your system, you do so at your own risk.

If you experience any strange problems during the installation of MySQL, or with some common utilities hanging, it is very likely that they are either library or compiler related. If this is the case, using our binary resolves them.

If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

This problem can be avoided by one of the following methods:

- Link clients with the `-Wl,r/full/path/to/libmysqlclient.so` flag rather than with `-Lpath`).
- Copy `libmysqlclient.so` to `/usr/lib`.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you are using the Fujitsu compiler (`fcc/FCC`), you may have some problems compiling MySQL because the Linux header files are very `gcc` oriented. The following `configure` line should work with `fcc/FCC`:

```
CC=fcc CFLAGS="-O -K fast -K lib -K omitfp -Kpreex -D_GNU_SOURCE \
-DCONST=const -DNO_STRTOLL_PROTO" \
CXX=fcc CXXFLAGS="-O -K fast -K lib \
-K omitfp -K preex --no_exceptions --no_rtti -D_GNU_SOURCE \
-DCONST=const -Dalloca=__builtin_alloca -DNO_STRTOLL_PROTO \
'-D_EXTERN_INLINE=static __inline'" \
./configure \
--prefix=/usr/local/mysql --enable-asm \
--with-mysqld-ldflags=-all-static --disable-shared \
--with-low-memory
```


2.13.1.4. Linux Post-Installation Notes

`mysql.server` can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).

If MySQL cannot open enough files or connections, it may be that you have not configured Linux to handle enough files.

In Linux 2.2 and onward, you can check the number of allocated file handles as follows:

```
shell> cat /proc/sys/fs/file-max
shell> cat /proc/sys/fs/dquot-max
shell> cat /proc/sys/fs/super-max
```

If you have more than 16MB of memory, you should add something like the following to your init scripts (for example, `/etc/init.d/boot.local` on SuSE Linux):

```
echo 65536 > /proc/sys/fs/file-max
echo 8192 > /proc/sys/fs/dquot-max
echo 1024 > /proc/sys/fs/super-max
```

You can also run the `echo` commands from the command line as `root`, but these settings are lost the next time your computer restarts.

Alternatively, you can set these parameters on startup by using the `sysctl` tool, which is used by many Linux distributions (including SuSE Linux 8.0 and later). Put the following values into a file named `/etc/sysctl.conf`:

```
# Increase some values for MySQL
fs.file-max = 65536
fs.dquot-max = 8192
fs.super-max = 1024
```

You should also add the following to `/etc/my.cnf`:

```
[mysqld_safe]
open-files-limit=8192
```

This should allow the server a limit of 8,192 for the combined number of connections and open files.

The `STACK_SIZE` constant in LinuxThreads controls the spacing of thread stacks in the address space. It needs to be large enough so that there is plenty of room for each individual thread stack, but small enough to keep the stack of some threads from running into the global `mysqld` data. Unfortunately, as we have experimentally discovered, the Linux implementation of `mmap()` successfully unmaps a mapped region if you ask it to map out an address currently in use, zeroing out the data on the entire page instead of returning an error. So, the safety of `mysqld` or any other threaded application depends on the “gentlemanly” behavior of the code that creates threads. The user must take measures to make sure that the number of running threads at any given time is sufficiently low for thread stacks to stay away from the global heap. With `mysqld`, you should enforce this behavior by setting a reasonable value for the `max_connections` variable.

If you build MySQL yourself, you can patch LinuxThreads for better stack use. See [Section 2.13.1.3, “Linux Source Distribution Notes”](#). If you do not want to patch LinuxThreads, you should set `max_connections` to a value no higher than 500. It should be even less if you have a large key buffer, large heap tables, or some other things that make `mysqld` allocate a lot of memory, or if you are running a 2.2 kernel with a 2GB patch. If you are using our binary or RPM version, you can safely set `max_connections` at 1500, assuming no large key buffer or heap tables with lots of data. The more you reduce `STACK_SIZE` in LinuxThreads the more threads you can safely create. We recommend values between 128KB and 256KB.

If you use a lot of concurrent connections, you may suffer from a “feature” in the 2.2 kernel that attempts to prevent fork bomb attacks by penalizing a process for forking or cloning a child. This causes MySQL not to scale well as you increase the number of concurrent clients. On single-CPU systems, we have seen this manifest as very slow thread creation; it may take a long time to connect to MySQL (as long as one minute), and it may take just as long to shut it down. On multiple-CPU systems, we have observed a gradual drop in query speed as the number of clients increases. In the process of trying to find a solution, we have received a kernel patch from one of our users who claimed it helped for his site. This patch is available at <http://dev.mysql.com/Downloads/Patches/linux-fork.patch>. We have done rather extensive testing of this patch on both development and production systems. It has significantly improved MySQL performance without causing any problems and we recommend it to our users who still run high-load servers on 2.2 kernels.

This issue has been fixed in the 2.4 kernel, so if you are not satisfied with the current performance of your system, rather than patching your 2.2 kernel, it might be easier to upgrade to 2.4. On SMP systems, upgrading also gives you a nice SMP boost in addition to fixing

the fairness bug.

We have tested MySQL on the 2.4 kernel on a two-CPU machine and found MySQL scales *much* better. There was virtually no slowdown on query throughput all the way up to 1,000 clients, and the MySQL scaling factor (computed as the ratio of maximum throughput to the throughput for one client) was 180%. We have observed similar results on a four-CPU system: Virtually no slowdown as the number of clients was increased up to 1,000, and a 300% scaling factor. Based on these results, for a high-load SMP server using a 2.2 kernel, we definitely recommend upgrading to the 2.4 kernel at this point.

We have discovered that it is essential to run the `mysqld` process with the highest possible priority on the 2.4 kernel to achieve maximum performance. This can be done by adding a `renice -20 $$` command to `mysqld_safe`. In our testing on a four-CPU machine, increasing the priority resulted in a 60% throughput increase with 400 clients.

We are currently also trying to collect more information on how well MySQL performs with a 2.4 kernel on four-way and eight-way systems. If you have access such a system and have done some benchmarks, please send an email message to [<benchmarks@mysql.com>](mailto:benchmarks@mysql.com) with the results. We will review them for inclusion in the manual.

If you see a dead `mysqld` server process with `ps`, this usually means that you have found a bug in MySQL or you have a corrupted table. See [Section B.1.4.2, “What to Do If MySQL Keeps Crashing”](#).

To get a core dump on Linux if `mysqld` dies with a `SIGSEGV` signal, you can start `mysqld` with the `--core-file` option. Note that you also probably need to raise the core file size by adding `ulimit -c 1000000` to `mysqld_safe` or starting `mysqld_safe` with `--core-file-size=1000000`. See [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#).

2.13.1.5. Linux x86 Notes

MySQL requires `libc` 5.4.12 or newer. It is known to work with `libc` 5.4.46. `glibc` 2.0.6 and later should also work. There have been some problems with the `glibc` RPMs from Red Hat, so if you have problems, check whether there are any updates. The `glibc` 2.0.7-19 and 2.0.7-29 RPMs are known to work.

If you are using Red Hat 8.0 or a new `glibc` 2.2.x library, you may see `mysqld` die in `gethostbyaddr()`. This happens because the new `glibc` library requires a stack size greater than 128KB for this call. To fix the problem, start `mysqld` with the `-thread-stack=192K` option. (Use `-O thread_stack=192K` before MySQL 4.) This stack size is the default on MySQL 4.0.10 and above, so you should not see the problem.

If you are using `gcc` 3.0 and above to compile MySQL, you must install the `libstdc++v3` library before compiling MySQL; if you don't do this, you get an error about a missing `__cxa_pure_virtual` symbol during linking.

On some older Linux distributions, `configure` may produce an error like this:

```
Syntax error in sched.h. Change _P to __P in the
/usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

Just do what the error message says. Add an extra underscore to the `_P` macro name that has only one underscore, and then try again.

You may get some warnings when compiling. Those shown here can be ignored:

```
mysqld.cc -o objs-thread/mysqld.o
mysqld.cc: In function `void init_signals()':
mysqld.cc:315: warning: assignment of negative value `-1' to
`long unsigned int'
mysqld.cc: In function `void * signal_hand(void *)':
mysqld.cc:346: warning: assignment of negative value `-1' to
`long unsigned int'
```

If `mysqld` always dumps core when it starts, the problem may be that you have an old `/lib/libc.a`. Try renaming it, and then remove `sql/mysqld` and do a new `make install` and try again. This problem has been reported on some Slackware installations.

If you get the following error when linking `mysqld`, it means that your `libg++.a` is not installed correctly:

```
/usr/lib/libc.a(putc.o): In function `_IO_putc':
putc.o(.text+0x0): multiple definition of `_IO_putc'
```

You can avoid using `libg++.a` by running `configure` like this:

```
shell> CXX=gcc ./configure
```


2.13.1.6. Linux SPARC Notes

In some implementations, `readdir_r()` is broken. The symptom is that the `SHOW DATABASES` statement always returns an empty set. This can be fixed by removing `HAVE_READDIR_R` from `config.h` after configuring and before compiling.

2.13.1.7. Linux Alpha Notes

We have tested MySQL 5.1 on Alpha with our benchmarks and test suite, and it appears to work well.

We currently build the MySQL binary packages on SuSE Linux 7.0 for AXP, kernel 2.4.4-SMP, Compaq C compiler (V6.2-505) and Compaq C++ compiler (V6.3-006) on a Compaq DS20 machine with an Alpha EV6 processor.

You can find the preceding compilers at <http://www.support.compaq.com/alpha-tools/>. By using these compilers rather than `gcc`, we get about 9-14% better MySQL performance.

For MySQL on Alpha, we use the `-arch generic` flag to our compile options, which ensures that the binary runs on all Alpha processors. We also compile statically to avoid library problems. The `configure` command looks like this:

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx \  
CXXFLAGS="-fast -arch generic -noexceptions -nortti" \  
./configure --prefix=/usr/local/mysql --disable-shared \  
--with-extra-charsets=complex --enable-thread-safe-client \  
--with-mysqld-ldflags=-non_shared --with-client-ldflags=-non_shared
```

If you want to use `egcs`, the following `configure` line worked for us:

```
CFLAGS="-O3 -fomit-frame-pointer" CXX=gcc \  
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \  
-fno-exceptions -fno-rtti" \  
./configure --prefix=/usr/local/mysql --disable-shared
```

Some known problems when running MySQL on Linux-Alpha:

- Debugging threaded applications like MySQL does not work with `gdb` 4.18. You should use `gdb` 5.1 instead.
- If you try linking `mysqld` statically when using `gcc`, the resulting image dumps core at startup time. In other words, *do not* use `-with-mysqld-ldflags=-all-static` with `gcc`.

2.13.1.8. Linux PowerPC Notes

MySQL should work on MkLinux with the newest `glibc` package (tested with `glibc` 2.0.7).

2.13.1.9. Linux MIPS Notes

To get MySQL to work on Qube2 (Linux Mips), you need the newest `glibc` libraries. `glibc-2.0.7-29C2` is known to work. You must also use the `egcs` C++ compiler (`egcs` 1.0.2-9, `gcc` 2.95.2 or newer).

2.13.1.10. Linux IA-64 Notes

To get MySQL to compile on Linux IA-64, we use the following `configure` command for building with `gcc` 2.96:

```
CC=gcc \  
CFLAGS="-O3 -fno-omit-frame-pointer" \  
CXX=gcc \  
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \  
-fno-exceptions -fno-rtti" \  
./configure --prefix=/usr/local/mysql \  
"--with-comment=Official MySQL binary" \  
--with-extra-charsets=complex
```

On IA-64, the MySQL client binaries use shared libraries. This means that if you install our binary distribution at a location other than `/usr/local/mysql`, you need to add the path of the directory where you have `libmysqlclient.so` installed either to the `/etc/ld.so.conf` file or to the value of your `LD_LIBRARY_PATH` environment variable.

See [Section B.1.3.1, "Problems Linking to the MySQL Client Library"](#).

2.13.1.11. SELinux Notes

RHEL4 comes with SELinux, which supports tighter access control for processes. If SELinux is enabled (`SELINUX` in `/etc/selinux/config` is set to `enforcing`, `SELINUXTYPE` is set to either `targeted` or `strict`), you might encounter problems installing MySQL AB RPM packages.

Red Hat has an update that solves this. It involves an update of the “security policy” specification to handle the install structure of the RPMs provided by MySQL AB. For further information, see https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=167551 and <http://rhn.redhat.com/errata/RHBA-2006-0049.html>.

The preceding discussion applies only to RHEL4. The patch is unnecessary for RHEL5.

2.13.2. Mac OS X Notes

On Mac OS X, `tar` cannot handle long filenames. If you need to unpack a `.tar.gz` distribution, use `gnutar` instead.

2.13.2.1. Mac OS X 10.x (Darwin)

MySQL should work without major problems on Mac OS X 10.x (Darwin).

Known issues:

- If you have problems with performance under heavy load, try using the `--skip-thread-priority` option to `mysqld`. This runs all threads with the same priority. On Mac OS X, this gives better performance, at least until Apple fixes its thread scheduler.
- The connection times (`wait_timeout`, `interactive_timeout` and `net_read_timeout`) values are not honored.

This is probably a signal handling problem in the thread library where the signal doesn't break a pending read and we hope that a future update to the thread libraries will fix this.

Our binary for Mac OS X is compiled on Darwin 6.3 with the following `configure` line:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc \  
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \  
-fno-exceptions -fno-rtti" \  
./configure --prefix=/usr/local/mysql \  
--with-extra-charsets=complex --enable-thread-safe-client \  
--enable-local-infile --disable-shared
```

See [Section 2.5, “Installing MySQL on Mac OS X”](#).

2.13.2.2. Mac OS X Server 1.2 (Rhapsody)

For current versions of Mac OS X Server, no operating system changes are necessary before compiling MySQL. Compiling for the Server platform is the same as for the client version of Mac OS X.

For older versions (Mac OS X Server 1.2, a.k.a. Rhapsody), you must first install a pthread package before trying to configure MySQL.

See [Section 2.5, “Installing MySQL on Mac OS X”](#).

2.13.3. Solaris Notes

For information about installing MySQL on Solaris using PKG distributions, see [Section 2.6, “Installing MySQL on Solaris”](#).

On Solaris, you may run into trouble even before you get the MySQL distribution unpacked, as the Solaris `tar` cannot handle long filenames. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution. You can find a precompiled copy for Solaris at <http://dev.mysql.com/downloads/os-solaris.html>.

Sun native threads work only on Solaris 2.5 and higher. For Solaris 2.4 and earlier, MySQL automatically uses MIT-pthreads. See [Section 2.9.5, “MIT-pthreads Notes”](#).

If you get the following error from `configure`, it means that you have something wrong with your compiler installation:

```
checking for restartable system calls... configure: error can not
run test programs while cross compiling
```

In this case, you should upgrade your compiler to a newer version. You may also be able to solve this problem by inserting the following row into the `config.cache` file:

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

If you are using Solaris on a SPARC, the recommended compiler is `gcc` 2.95.2 or 3.2. You can find this at <http://gcc.gnu.org/>. Note that `egcs` 1.1.1 and `gcc` 2.8.1 do not work reliably on SPARC.

The recommended `configure` line when using `gcc` 2.95.2 is:

```
CC=gcc CFLAGS="-O3" \
CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory \
--enable-assembler
```

If you have an UltraSPARC system, you can get 4% better performance by adding `-mcpu=v8 -wa, -xarch=v8plusa` to the `CFLAGS` and `CXXFLAGS` environment variables.

If you have Sun's Forte 5.0 (or newer) compiler, you can run `configure` like this:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt" \
CXX=CC CXXFLAGS="-noex -mt" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

To create a 64-bit binary with Sun's Forte compiler, use the following configuration options:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt -xarch=v9" \
CXX=CC CXXFLAGS="-noex -mt -xarch=v9" ASFLAGS="-xarch=v9" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

To create a 64-bit Solaris binary using `gcc`, add `-m64` to `CFLAGS` and `CXXFLAGS` and remove `--enable-assembler` from the `configure` line.

In the MySQL benchmarks, we obtained a 4% speed increase on UltraSPARC when using Forte 5.0 in 32-bit mode, as compared to using `gcc` 3.2 with the `-mcpu` flag.

If you create a 64-bit `mysqld` binary, it is 4% slower than the 32-bit binary, but can handle more threads and memory.

When using Solaris 10 for x86_64, you should mount any filesystems on which you intend to store `InnoDB` files with the `forcedir-ectio` option. (By default mounting is done without this option.) Failing to do so will cause a significant drop in performance when using the `InnoDB` storage engine on this platform.

If you get a problem with `fdatasync` or `sched_yield`, you can fix this by adding `LIBS=-lrt` to the `configure` line

For compilers older than WorkShop 5.3, you might have to edit the `configure` script. Change this line:

```
#if !defined(__STDC__) || __STDC__ != 1
```

To this:

```
#if !defined(__STDC__)
```

If you turn on `__STDC__` with the `-Xc` option, the Sun compiler can't compile with the Solaris `pthread.h` header file. This is a Sun bug (broken compiler or broken include file).

If `mysqld` issues the following error message when you run it, you have tried to compile MySQL with the Sun compiler without enabling the `-mt` multi-thread option:

```
libc internal error: _rmutex_unlock: rmutex not held
```

Add `-mt` to `CFLAGS` and `CXXFLAGS` and recompile.

If you are using the SFW version of `gcc` (which comes with Solaris 8), you must add `/opt/sfw/lib` to the environment variable `LD_LIBRARY_PATH` before running `configure`.

If you are using the `gcc` available from sunfreeware.com, you may have many problems. To avoid this, you should recompile `gcc` and GNU `binutils` on the machine where you are running them.

If you get the following error when compiling MySQL with `gcc`, it means that your `gcc` is not configured for your version of Solaris:

```
shell> gcc -O3 -g -O2 -DDEBUG_OFF -o thr_alarm ...
./thr_alarm.c: In function `signal_hand':
./thr_alarm.c:556: too many arguments to function `sigwait'
```

The proper thing to do in this case is to get the newest version of `gcc` and compile it with your current `gcc` compiler. At least for Solaris 2.5, almost all binary versions of `gcc` have old, unusable include files that break all programs that use threads, and possibly other programs as well.

Solaris does not provide static versions of all system libraries (`libpthreads` and `libdl`), so you cannot compile MySQL with `-static`. If you try to do so, you get one of the following errors:

```
ld: fatal: library -ldl: not found
undefined reference to `dlopen'
cannot find -lrt
```

If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

This problem can be avoided by one of the following methods:

- Link clients with the `-Wl,r/full/path/to/libmysqlclient.so` flag rather than with `-Lpath`).
- Copy `libmysqlclient.so` to `/usr/lib`.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you have problems with `configure` trying to link with `-lz` when you don't have `zlib` installed, you have two options:

- If you want to be able to use the compressed communication protocol, you need to get and install `zlib` from ftp.gnu.org.
- Run `configure` with the `--with-named-z-libs=no` option when building MySQL.

If you are using `gcc` and have problems with loading user-defined functions (UDFs) into MySQL, try adding `-lgcc` to the link line for the UDF.

If you would like MySQL to start automatically, you can copy `support-files/mysql.server` to `/etc/init.d` and create a symbolic link to it named `/etc/rc3.d/S99mysql.server`.

If too many processes try to connect very rapidly to `mysqld`, you should see this error in the MySQL log:

```
Error in accept: Protocol error
```

You might try starting the server with the `--back_log=50` option as a workaround for this. (Use `-O back_log=50` before MySQL 4.)

Solaris doesn't support core files for `setuid()` applications, so you can't get a core file from `mysqld` if you are using the `--user` option.

2.13.3.1. Solaris 2.7/2.8 Notes

Normally, you can use a Solaris 2.6 binary on Solaris 2.7 and 2.8. Most of the Solaris 2.6 issues also apply for Solaris 2.7 and 2.8.

MySQL should be able to detect new versions of Solaris automatically and enable workarounds for the following problems.

Solaris 2.7 / 2.8 has some bugs in the include files. You may see the following error when you use `gcc`:

```
/usr/include/widec.h:42: warning: `getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

If this occurs, you can fix the problem by copying `/usr/include/widec.h` to `.../lib/gcc-lib/os/gcc-version/include` and changing line 41 from this:

```
#if !defined(lint) && !defined(__lint)
```

To this:

```
#if !defined(lint) && !defined(__lint) && !defined(getwc)
```

Alternatively, you can edit `/usr/include/widec.h` directly. Either way, after you make the fix, you should remove `config.cache` and run `configure` again.

If you get the following errors when you run `make`, it's because `configure` didn't detect the `curses.h` file (probably because of the error in `/usr/include/widec.h`):

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before `;'
/usr/include/term.h:1081: syntax error before `;'
```

The solution to this problem is to do one of the following:

- Configure with `CFLAGS=-DHAVE_CURSES_H CXXFLAGS=-DHAVE_CURSES_H ./configure`.
- Edit `/usr/include/widec.h` as indicated in the preceding discussion and re-run `configure`.
- Remove the `#define HAVE_TERM` line from the `config.h` file and run `make` again.

If your linker cannot find `-lz` when linking client programs, the problem is probably that your `libz.so` file is installed in `/usr/local/lib`. You can fix this problem by one of the following methods:

- Add `/usr/local/lib` to `LD_LIBRARY_PATH`.
- Add a link to `libz.so` from `/lib`.
- If you are using Solaris 8, you can install the optional `zlib` from your Solaris 8 CD distribution.
- Run `configure` with the `--with-named-z-libs=no` option when building MySQL.

2.13.3.2. Solaris x86 Notes

On Solaris 8 on x86, `mysqld` dumps core if you remove the debug symbols using `strip`.

If you are using `gcc` or `egcs` on Solaris x86 and you experience problems with core dumps under load, you should use the following `configure` command:

```
CC=gcc CFLAGS="-O3 -fomit-frame-pointer -DHAVE_CURSES_H" \
CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \
-fno-exceptions -fno-rtti -DHAVE_CURSES_H" \
./configure --prefix=/usr/local/mysql
```

This avoids problems with the `libstdc++` library and with C++ exceptions.

If this doesn't help, you should compile a debug version and run it with a trace file or under [gdb](#). See [MySQL Internals: Porting](#).

2.13.4. BSD Notes

This section provides information about using MySQL on variants of BSD Unix.

2.13.4.1. FreeBSD Notes

FreeBSD 4.x or newer is recommended for running MySQL, because the thread package is much more integrated. To get a secure and stable system, you should use only FreeBSD kernels that are marked **-RELEASE**.

The easiest (and preferred) way to install MySQL is to use the [mysql-server](#) and [mysql-client](#) ports available at <http://www.freebsd.org/>. Using these ports gives you the following benefits:

- A working MySQL with all optimizations enabled that are known to work on your version of FreeBSD.
- Automatic configuration and build.
- Startup scripts installed in `/usr/local/etc/rc.d`.
- The ability to use `pkg_info -L` to see which files are installed.
- The ability to use `pkg_delete` to remove MySQL if you no longer want it on your machine.

It is recommended you use MIT-pthreads on FreeBSD 2.x, and native threads on FreeBSD 3 and up. It is possible to run with native threads on some late 2.2.x versions, but you may encounter problems shutting down `mysqld`.

Unfortunately, certain function calls on FreeBSD are not yet fully thread-safe. Most notably, this includes the `gethostbyname()` function, which is used by MySQL to convert hostnames into IP addresses. Under certain circumstances, the `mysqld` process suddenly causes 100% CPU load and is unresponsive. If you encounter this problem, try to start MySQL using the `--skip-name-resolve` option.

Alternatively, you can link MySQL on FreeBSD 4.x against the LinuxThreads library, which avoids a few of the problems that the native FreeBSD thread implementation has. For a very good comparison of LinuxThreads versus native threads, see Jeremy Zawodny's article *FreeBSD or Linux for your MySQL Server?* at <http://jeremy.zawodny.com/blog/archives/000697.html>.

Known problem when using LinuxThreads on FreeBSD is:

- The connection times (`wait_timeout`, `interactive_timeout` and `net_read_timeout`) values are not honored. The symptom is that persistent connections can hang for a very long time without getting closed down and that a 'kill' for a thread will not take affect until the thread does it a new command

This is probably a signal handling problem in the thread library where the signal doesn't break a pending read. This is supposed to be fixed in FreeBSD 5.0

The MySQL build process requires GNU make ([gmake](#)) to work. If GNU [make](#) is not available, you must install it first before compiling MySQL.

The recommended way to compile and install MySQL on FreeBSD with [gcc](#) (2.95.2 and up) is:

```
CC=gcc CFLAGS="-O2 -fno-strength-reduce" \  
CXX=gcc CXXFLAGS="-O2 -fno-rtti -fno-exceptions \  
-felide-constructors -fno-strength-reduce" \  
./configure --prefix=/usr/local/mysql --enable-assembly  
gmake  
gmake install  
cd /usr/local/mysql  
bin/mysql_install_db --user=mysql  
bin/mysqld_safe &
```

If you notice that [configure](#) uses MIT-pthreads, you should read the MIT-pthreads notes. See [Section 2.9.5, "MIT-pthreads Notes"](#).

If you get an error from `make install` that it can't find `/usr/include/pthreads`, [configure](#) didn't detect that you need

MIT-pthreads. To fix this problem, remove `config.cache`, and then re-run `configure` with the `--with-mit-threads` option.

Be sure that your name resolver setup is correct. Otherwise, you may experience resolver delays or failures when connecting to `mysqld`. Also make sure that the `localhost` entry in the `/etc/hosts` file is correct. The file should start with a line similar to this:

```
127.0.0.1      localhost localhost.your.domain
```

FreeBSD is known to have a very low default file handle limit. See [Section B.1.2.17, “'FILE' NOT FOUND and Similar Errors](#)”. Start the server by using the `--open-files-limit` option for `mysqld_safe`, or raise the limits for the `mysqld` user in `/etc/login.conf` and rebuild it with `cap_mkdb /etc/login.conf`. Also be sure that you set the appropriate class for this user in the password file if you are not using the default (use `chpass mysqld-user-name`). See [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script](#)”.

FreeBSD limits the size of a process to 512MB, even if you have much more RAM available on the system. So you may get an error such as this:

```
Out of memory (Needed 16391 bytes)
```

In current versions of FreeBSD (at least 4.x and greater), you may increase this limit by adding the following entries to the `/boot/loader.conf` file and rebooting the machine (these are not settings that can be changed at run time with the `sysctl` command):

```
kern.maxdsiz="1073741824" # 1GB
kern.dflsiz="1073741824" # 1GB
kern.maxssiz="134217728" # 128MB
```

For older versions of FreeBSD, you must recompile your kernel to change the maximum data segment size for a process. In this case, you should look at the `MAXDSIZ` option in the `LINT` config file for more information.

If you get problems with the current date in MySQL, setting the `TZ` variable should help. See [Section 2.14, “Environment Variables](#)”.

2.13.4.2. NetBSD Notes

To compile on NetBSD, you need GNU `make`. Otherwise, the build process fails when `make` tries to run `lint` on C++ files.

2.13.4.3. OpenBSD 2.5 Notes

On OpenBSD 2.5, you can compile MySQL with native threads with the following options:

```
CFLAGS=-pthread CXXFLAGS=-pthread ./configure --with-mit-threads=no
```

2.13.4.4. BSD/OS Version 2.x Notes

If you get the following error when compiling MySQL, your `ulimit` value for virtual memory is too low:

```
item_func.h: In method
`Item_func_ge::Item_func_ge(const Item_func_ge &)':
item_func.h:28: virtual memory exhausted
make[2]: *** [item_func.o] Error 1
```

Try using `ulimit -v 80000` and run `make` again. If this doesn't work and you are using `bash`, try switching to `csch` or `sh`; some BSDI users have reported problems with `bash` and `ulimit`.

If you are using `gcc`, you may also have to use the `--with-low-memory` flag for `configure` to be able to compile `sql_yacc.cc`.

If you get problems with the current date in MySQL, setting the `TZ` variable should help. See [Section 2.14, “Environment Variables](#)”.

2.13.4.5. BSD/OS Version 3.x Notes

Upgrade to BSD/OS 3.1. If that is not possible, install BSDIpatch M300-038.

Use the following command when configuring MySQL:

```
env CXX=shlcc++ CC=shlcc2 \  
./configure \  
  --prefix=/usr/local/mysql \  
  --localstatedir=/var/mysql \  
  --without-perl \  
  --with-unix-socket-path=/var/mysql/mysql.sock
```

The following is also known to work:

```
env CC=gcc CXX=gcc CXXFLAGS=-O3 \  
./configure \  
  --prefix=/usr/local/mysql \  
  --with-unix-socket-path=/var/mysql/mysql.sock
```

You can change the directory locations if you wish, or just use the defaults by not specifying any locations.

If you have problems with performance under heavy load, try using the `--skip-thread-priority` option to `mysqld`. This runs all threads with the same priority. On BSDI 3.1, this gives better performance, at least until BSDI fixes its thread scheduler.

If you get the error `virtual memory exhausted` while compiling, you should try using `ulimit -v 80000` and running `make` again. If this doesn't work and you are using `bash`, try switching to `csh` or `sh`; some BSDI users have reported problems with `bash` and `ulimit`.

2.13.4.6. BSD/OS Version 4.x Notes

BSDI 4.x has some thread-related bugs. If you want to use MySQL on this, you should install all thread-related patches. At least M400-023 should be installed.

On some BSDI 4.x systems, you may get problems with shared libraries. The symptom is that you can't execute any client programs, for example, `mysqladmin`. In this case, you need to reconfigure not to use shared libraries with the `--disable-shared` option to configure.

Some customers have had problems on BSDI 4.0.1 that the `mysqld` binary after a while can't open tables. This occurs because some library/system-related bug causes `mysqld` to change current directory without having asked for that to happen.

The fix is to either upgrade MySQL to at least version 3.23.34 or, after running `configure`, remove the line `#define HAVE_REALPATH` from `config.h` before running `make`.

Note that this means that you can't symbolically link a database directories to another database directory or symbolic link a table to another database on BSDI. (Making a symbolic link to another disk is okay).

2.13.5. Other Unix Notes

2.13.5.1. HP-UX Version 10.20 Notes

There are a couple of small problems when compiling MySQL on HP-UX. We recommend that you use `gcc` instead of the HP-UX native compiler, because `gcc` produces better code.

We recommend using `gcc` 2.95 on HP-UX. Don't use high optimization flags (such as `-O6`) because they may not be safe on HP-UX.

The following `configure` line should work with `gcc` 2.95:

```
CFLAGS="-I/opt/dce/include -fpic" \  
CXXFLAGS="-I/opt/dce/include -felide-constructors -fno-exceptions \  
-fno-rtti" \  
CXX=gcc \  
./configure --with-pthread \  
  --with-named-thread-libs='-ldce' \  
  --prefix=/usr/local/mysql --disable-shared
```

The following `configure` line should work with `gcc` 3.1:

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc \  
CXXFLAGS="-DHPUX -I/opt/dce/include -felide-constructors \  
-fno-exceptions -fno-rtti -O3 -fPIC" \  
./configure --prefix=/usr/local/mysql \  
  --with-extra-charsets=complex --enable-thread-safe-client \  
  --enable-local-infile --with-pthread \  
  --with-named-thread-libs=-ldce --with-lib-ccflags=-fPIC
```



```
--disable-shared
```

2.13.5.2. HP-UX Version 11.x Notes

Because of some critical bugs in the standard HP-UX libraries, you should install the following patches before trying to run MySQL on HP-UX 11.0:

```
PHKL_22840 Streams cumulative
PHNE_22397 ARPA cumulative
```

This solves the problem of getting `EWOULDBLOCK` from `recv()` and `EBADF` from `accept()` in threaded applications.

If you are using `gcc` 2.95.1 on an unpatched HP-UX 11.x system, you may get the following error:

```
In file included from /usr/include/unistd.h:11,
                 from ../include/global.h:125,
                 from mysql_priv.h:15,
                 from item.cc:19:
/usr/include/sys/unistd.h:184: declaration of C function ...
/usr/include/sys/unistd.h:184: previous declaration ...
In file included from item.h:306,
                 from mysql_priv.h:158,
                 from item.cc:19:
```

The problem is that HP-UX does not define `pthread_atfork()` consistently. It has conflicting prototypes in `/usr/include/sys/unistd.h:184` and `/usr/include/sys/unistd.h:440`.

One solution is to copy `/usr/include/sys/unistd.h` into `mysql/include` and edit `unistd.h` and change it to match the definition in `pthread.h`. Look for this line:

```
extern int pthread_atfork(void (*prepare)(), void (*parent)(),
                        void (*child)());
```

Change it to look like this:

```
extern int pthread_atfork(void (*prepare)(void), void (*parent)(void),
                        void (*child)(void));
```

After making the change, the following `configure` line should work:

```
CFLAGS="-fomit-frame-pointer -O3 -fpic" CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -O3" \
./configure --prefix=/usr/local/mysql --disable-shared
```

If you are using HP-UX compiler, you can use the following command (which has been tested with `cc` B.11.11.04):

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure \
--with-extra-character-set=complex
```

You can ignore any errors of the following type:

```
aCC: warning 901: unknown option: '-3': use +help for online
documentation
```

If you get the following error from `configure`, verify that you don't have the path to the K&R compiler before the path to the HP-UX C and C++ compiler:

```
checking for cc option to accept ANSI C... no
configure: error: MySQL requires an ANSI C compiler (and a C++ compiler).
Try gcc. See the Installation chapter in the Reference Manual.
```

Another reason for not being able to compile is that you didn't define the `+DD64` flags as just described.

Another possibility for HP-UX 11 is to use the MySQL binaries provided at <http://dev.mysql.com/downloads/>, which we have built and tested ourselves. We have also received reports that the HP-UX 10.20 binaries supplied by MySQL can be run successfully on HP-UX 11. If you encounter problems, you should be sure to check your HP-UX patch level.

2.13.5.3. IBM-AIX notes

Automatic detection of `xlc` is missing from Autoconf, so a number of variables need to be set before running `configure`. The following example uses the IBM compiler:

```
export CC="xlc_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192"
export CXX="xlc_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192"
export CFLAGS="-I /usr/local/include"
export LDFLAGS="-L /usr/local/lib"
export CPPFLAGS=$CFLAGS
export CXXFLAGS=$CFLAGS

./configure --prefix=/usr/local \
    --localstatedir=/var/mysql \
    --sbindir='/usr/local/bin' \
    --libexecdir='/usr/local/bin' \
    --enable-thread-safe-client \
    --enable-large-files
```

The preceding options are used to compile the MySQL distribution that can be found at <http://www-frec.bull.com/>.

If you change the `-O3` to `-O2` in the preceding `configure` line, you must also remove the `-qstrict` option. This is a limitation in the IBM C compiler.

If you are using `gcc` or `egcs` to compile MySQL, you *must* use the `-fno-exceptions` flag, because the exception handling in `gcc/egcs` is not thread-safe! (This is tested with `egcs` 1.1.) There are also some known problems with IBM's assembler that may cause it to generate bad code when used with `gcc`.

We recommend the following `configure` line with `egcs` and `gcc` 2.95 on AIX:

```
CC="gcc -pipe -mcpu=power -wa,-many" \
CXX="gcc -pipe -mcpu=power -wa,-many" \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory
```

The `-wa, -many` option is necessary for the compile to be successful. IBM is aware of this problem but is in no hurry to fix it because of the workaround that is available. We don't know if the `-fno-exceptions` is required with `gcc` 2.95, but because MySQL doesn't use exceptions and the option generates faster code, we recommend that you should always use it with `egcs` / `gcc`.

If you get a problem with assembler code, try changing the `-mcpu=xxx` option to match your CPU. Typically `power2`, `power`, or `powerpc` may need to be used. Alternatively, you might need to use `604` or `604e`. We are not positive but suspect that `power` would likely be safe most of the time, even on a `power2` machine.

If you don't know what your CPU is, execute a `uname -m` command. It produces a string that looks like `000514676700`, with a format of `xyyyyyymmss` where `xx` and `ss` are always `00`, `yyyyyy` is a unique system ID and `mm` is the ID of the CPU Planar. A chart of these values can be found at http://www16.boulder.ibm.com/pseries/en_US/cmds/aixcmds5/uname.htm.

This gives you a machine type and a machine model you can use to determine what type of CPU you have.

If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring as follows:

```
CFLAGS=-DDONT_USE_THR_ALARM CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti" \
-DDONT_USE_THR_ALARM \
./configure --prefix=/usr/local/mysql --with-debug \
    --with-low-memory
```

This doesn't affect the performance of MySQL, but has the side effect that you can't kill clients that are "sleeping" on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client dies when it issues its next command.

On some versions of AIX, linking with `libbind.a` makes `getservbyname()` dump core. This is an AIX bug and should be reported to IBM.

For AIX 4.2.1 and `gcc`, you have to make the following changes.

After configuring, edit `config.h` and `include/my_config.h` and change the line that says this:

```
#define HAVE_SNPRINTF 1
```

to this:

```
#undef HAVE_SNPRINTF
```

And finally, in `mysqld.cc`, you need to add a prototype for `initgroups()`.

```
#ifdef _AIX41
extern "C" int initgroups(const char *,int);
#endif
```

If you need to allocate a lot of memory to the `mysqld` process, it's not enough to just use `ulimit -d unlimited`. You may also have to modify `mysqld_safe` to add a line something like this:

```
export LDR_CNTRL='MAXDATA=0x80000000'
```

You can find more information about using a lot of memory at http://publib16.boulder.ibm.com/pseries/en_US/aixprgpd/genprogc/lrg_prg_support.htm.

Users of AIX 4.3 should use `gmake` instead of the `make` utility included with AIX.

As of AIX 4.1, the C compiler has been unbundled from AIX as a separate product. We recommend using `gcc` 3.3.2, which can be obtained here: <ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/RPMS/ppc/gcc/>

The steps for compiling MySQL on AIX with `gcc` 3.3.2 are similar to those for using `gcc` 2.95 (in particular, the need to edit `con-fig.h` and `my_config.h` after running `configure`). However, before running `configure`, you should also patch the `curses.h` file as follows:

```
/opt/freeware/lib/gcc-lib/powerpc-ibm-aix5.2.0.0/3.3.2/include/curses.h.ORIG
Mon Dec 26 02:17:28 2005
--- /opt/freeware/lib/gcc-lib/powerpc-ibm-aix5.2.0.0/3.3.2/include/curses.h
Mon Dec 26 02:40:13 2005
*****
*** 2023,2029 ***

    #endif /* _AIX32_CURSES */
! #if defined(__USE_FIXED_PROTOTYPES__) || defined(__cplusplus) || defined
(__STRICT_ANSI__)
    extern int delwin (WINDOW *);
    extern int endwin (void);
    extern int getcurx (WINDOW *);
--- 2023,2029 ----

    #endif /* _AIX32_CURSES */
! #if 0 && (defined(__USE_FIXED_PROTOTYPES__) || defined(__cplusplus)
|| defined
(__STRICT_ANSI__))
    extern int delwin (WINDOW *);
    extern int endwin (void);
    extern int getcurx (WINDOW *);
```

2.13.5.4. SunOS 4 Notes

On SunOS 4, MIT-pthreads is needed to compile MySQL. This in turn means you need GNU `make`.

Some SunOS 4 systems have problems with dynamic libraries and `libtool`. You can use the following `configure` line to avoid this problem:

```
./configure --disable-shared --with-mysqld-ldflags=-all-static
```

When compiling `readline`, you may get warnings about duplicate defines. These can be ignored.

When compiling `mysqld`, there are some `implicit declaration of function` warnings. These can be ignored.

2.13.5.5. Alpha-DEC-UNIX Notes (Tru64)

If you are using `egcs` 1.1.2 on Digital Unix, you should upgrade to `gcc` 2.95.2, because `egcs` on DEC has some serious bugs!

When compiling threaded programs under Digital Unix, the documentation recommends using the `-pthread` option for `cc` and `cxx`

and the `-lmach` `-lexc` libraries (in addition to `-lpthread`). You should run `configure` something like this:

```
CC="cc -pthread" CXX="c++ -pthread -O" \  
./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

When compiling `mysqld`, you may see a couple of warnings like this:

```
mysqld.cc: In function void handle_connections():  
mysqld.cc:626: passing long unsigned int '*' as argument 3 of  
accept(int,sockaddr *, int *)
```

You can safely ignore these warnings. They occur because `configure` can detect only errors, not warnings.

If you start the server directly from the command line, you may have problems with it dying when you log out. (When you log out, your outstanding processes receive a `SIGHUP` signal.) If so, try starting the server like this:

```
nohup mysqld [options] &
```

`nohup` causes the command following it to ignore any `SIGHUP` signal sent from the terminal. Alternatively, start the server by running `mysqld_safe`, which invokes `mysqld` using `nohup` for you. See [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#).

If you get a problem when compiling `mysys/get_opt.c`, just remove the `#define _NO_PROTO` line from the start of that file.

If you are using Compaq's CC compiler, the following `configure` line should work:

```
CC="cc -pthread"  
CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed \  
-speculate all -arch host"  
CXX="c++ -pthread"  
CXXFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed \  
-speculate all -arch host -noexceptions -nortti"  
export CC CFLAGS CXX CXXFLAGS  
./configure \  
--prefix=/usr/local/mysql \  
--with-low-memory \  
--enable-large-files \  
--enable-shared=yes \  
--with-named-thread-libs="-lpthread -lmach -lexc -lc"  
gnumake
```

If you get a problem with `libtool` when compiling with shared libraries as just shown, when linking `mysql`, you should be able to get around this by issuing these commands:

```
cd mysql  
/bin/sh ../libtool --mode=link c++ -pthread -O3 -DDEBUG_OFF \  
-O4 -ansi_alias -ansi_args -fast -inline speed \  
-speculate all -arch host -DUNDEF_HAVE_GETHOSTBYNAME_R \  
-o mysql.o readline.o sql_string.o completion_hash.o \  
../readline/libreadline.a -lcurses \  
../libmysql/.libs/libmysqlclient.so -lm  
cd ..  
gnumake  
gnumake install  
scripts/mysql_install_db
```

2.13.5.6. Alpha-DEC-OSF/1 Notes

If you have problems compiling and have DEC `CC` and `gcc` installed, try running `configure` like this:

```
CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \  
./configure --prefix=/usr/local/mysql
```

If you get problems with the `c_asm.h` file, you can create and use a 'dummy' `c_asm.h` file with:

```
touch include/c_asm.h  
CC=gcc CFLAGS=-I./include \  
CXX=gcc CXXFLAGS=-O3 \  
./configure --prefix=/usr/local/mysql
```

Note that the following problems with the `ld` program can be fixed by downloading the latest DEC (Compaq) patch kit from: [ht-](#)

[tp://ftp.support.compaq.com/public/unix/](http://ftp.support.compaq.com/public/unix/).

On OSF/1 V4.0D and compiler "DEC C V5.6-071 on Digital Unix V4.0 (Rev. 878)," the compiler had some strange behavior (undefined `asm` symbols). `/bin/ld` also appears to be broken (problems with `_exit undefined` errors occurring while linking `mysqld`). On this system, we have managed to compile MySQL with the following `configure` line, after replacing `/bin/ld` with the version from OSF 4.0C:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

With the Digital compiler "C++ V6.1-029," the following should work:

```
CC=cc -pthread
CFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \
    -speculate all -arch host
CXX=cxx -pthread
CXXFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \
    -speculate all -arch host -noexceptions -nortti
export CC CFLAGS CXX CXXFLAGS
./configure --prefix=/usr/mysql/mysql \
    --with-mysqld-ldflags=-all-static --disable-shared \
    --with-named-thread-libs="-lmach -lexc -lc"
```

In some versions of OSF/1, the `alloca()` function is broken. Fix this by removing the line in `config.h` that defines `'HAVE_ALLOCA'`.

The `alloca()` function also may have an incorrect prototype in `/usr/include/alloca.h`. This warning resulting from this can be ignored.

`configure` uses the following thread libraries automatically: `--with-named-thread-libs="-lpthread -lmach -lexc -lc"`.

When using `gcc`, you can also try running `configure` like this:

```
CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ...
```

If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring with:

```
CFLAGS=-DDONT_USE_THR_ALARM \
CXXFLAGS=-DDONT_USE_THR_ALARM \
./configure ...
```

This does not affect the performance of MySQL, but has the side effect that you can't kill clients that are "sleeping" on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client dies when it issues its next command.

With `gcc` 2.95.2, you may encounter the following compile error:

```
sql_acl.cc:1456: Internal compiler error in `scan_region',
at except.c:2566
Please submit a full bug report.
```

To fix this, you should change to the `sql` directory and do a cut-and-paste of the last `gcc` line, but change `-O3` to `-O0` (or add `-O0` immediately after `gcc` if you don't have any `-O` option on your compile line). After this is done, you can just change back to the top-level directory and run `make` again.

2.13.5.7. SGI Irix Notes

As of MySQL 5.0, we don't provide binaries for Irix any more.

If you are using Irix 6.5.3 or newer, `mysqld` is able to create threads only if you run it as a user that has `CAP_SCHED_MGT` privileges (such as `root`) or give the `mysqld` server this privilege with the following shell command:

```
chcap "CAP_SCHED_MGT+epi" /opt/mysql/libexec/mysqld
```

You may have to undefine some symbols in `config.h` after running `configure` and before compiling.

In some Irix implementations, the `alloca()` function is broken. If the `mysqld` server dies on some `SELECT` statements, remove the

lines from `config.h` that define `HAVE_ALLOC` and `HAVE_ALLOCA_H`. If `mysqladmin create` doesn't work, remove the line from `config.h` that defines `HAVE_READDIR_R`. You may have to remove the `HAVE_TERM_H` line as well.

SGI recommends that you install all the patches on this page as a set: http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

At the very minimum, you should install the latest kernel rollup, the latest `rld` rollup, and the latest `libc` rollup.

You definitely need all the POSIX patches on this page, for pthreads support:

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

If you get the something like the following error when compiling `mysql.cc`:

```
"usr/include/curses.h", line 82: error(1084):
invalid combination of type
```

Type the following in the top-level directory of your MySQL source tree:

```
extra/replace bool curses_bool < /usr/include/curses.h > include/curses.h
make
```

There have also been reports of scheduling problems. If only one thread is running, performance is slow. Avoid this by starting another client. This may lead to a two-to-tenfold increase in execution speed thereafter for the other thread. This is a poorly understood problem with Irix threads; you may have to improvise to find solutions until this can be fixed.

If you are compiling with `gcc`, you can use the following `configure` command:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql --enable-thread-safe-client \
--with-named-thread-libs=-lpthread
```

On Irix 6.5.11 with native Irix C and C++ compilers ver. 7.3.1.2, the following is reported to work

```
CC=cc CXX=CC CFLAGS='-O3 -n32 -TARGET:platform=IP22 -I/usr/local/include \
-L/usr/local/lib' CXXFLAGS='-O3 -n32 -TARGET:platform=IP22 \
-I/usr/local/include -L/usr/local/lib' \
./configure --prefix=/usr/local/mysql --with-innodb \
--with-libwrap=/usr/local \
--with-named-curses-libs=/usr/local/lib/libncurses.a
```

2.13.5.8. SCO UNIX and OpenServer 5.0.x Notes

The current port is tested only on `sc03.2v5.0.5`, `sc03.2v5.0.6`, and `sc03.2v5.0.7` systems. There has also been progress on a port to `sc03.2v4.2`. Open Server 5.0.8 (Legend) has native threads and allows files greater than 2GB. The current maximum file size is 2GB.

We have been able to compile MySQL with the following `configure` command on OpenServer with `gcc` 2.95.3.

```
CC=gcc CFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
CXX=gcc CXXFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
./configure --prefix=/usr/local/mysql \
--enable-thread-safe-client --with-innodb \
--with-openssl --with-vio --with-extra-charsets=complex
```

`gcc` is available at <ftp://ftp.sco.com/pub/openserver5/opensrc/gnutools-5.0.7Kj>.

This development system requires the OpenServer Execution Environment Supplement oss646B on OpenServer 5.0.6 and oss656B and The OpenSource libraries found in `gwxlibs`. All OpenSource tools are in the `opensrc` directory. They are available at <ftp://ftp.sco.com/pub/openserver5/opensrc/>.

We recommend using the latest production release of MySQL.

SCO provides operating system patches at <ftp://ftp.sco.com/pub/openserver5> for OpenServer 5.0.[0-6] and <ftp://ftp.sco.com/pub/openserverv5/507> for OpenServer 5.0.7.

SCO provides information about security fixes at <ftp://ftp.sco.com/pub/security/OpenServer> for OpenServer 5.0.x.

The maximum file size on an OpenSever 5.0.x system is 2GB.

The total memory which can be allocated for streams buffers, clists, and lock records cannot exceed 60MB on OpenServer 5.0.x.

Streams buffers are allocated in units of 4096 byte pages, clists are 70 bytes each, and lock records are 64 bytes each, so:

```
(NSTRPAGES × 4096) + (NCLIST × 70) + (MAX_FLCKREC × 64) <= 62914560
```

Follow this procedure to configure the Database Services option. If you are unsure whether an application requires this, see the documentation provided with the application.

1. Log in as `root`.
2. Enable the SUDS driver by editing the `/etc/conf/sdevice.d/suds` file. Change the `N` in the second field to a `Y`.
3. Use `mkdev aio` or the Hardware/Kernel Manager to enable support for asynchronous I/O and relink the kernel. To allow users to lock down memory for use with this type of I/O, update the `aiomemlock(F)` file. This file should be updated to include the names of users that can use AIO and the maximum amounts of memory they can lock down.
4. Many applications use `setuid` binaries so that you need to specify only a single user. See the documentation provided with the application to determine whether this is the case for your application.

After you complete this process, reboot the system to create a new kernel incorporating these changes.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

Value	Default	Min	Max
-----	-----	---	---
NBUF	0	24	450000
NHBUF	0	32	524288
NMPBUF	0	12	512
MAX_INODE	0	100	64000
MAX_FILE	0	100	64000
CTBUFSIZE	128	0	256
MAX_PROC	0	50	16000
MAX_REGION	0	500	160000
NCLIST	170	120	16640
MAXUP	100	15	16000
NOFILES	110	60	11000
NHINODE	128	64	8192
NAUTOUP	10	0	60
NGROUPS	8	0	128
BDFLUSHR	30	1	300
MAX_FLCKREC	0	50	16000
PUTBUFSZ	8000	2000	20000
MAXSLICE	100	25	100
ULIMIT	4194303	2048	4194303
* Streams Parameters			
NSTREAM	64	1	32768
NSTRPUSH	9	9	9
NMUXLINK	192	1	4096
STRMSGSZ	16384	4096	524288
STRCTLSZ	1024	1024	1024
STRMAXBLK	524288	4096	524288
NSTRPAGES	500	0	8000
STRSPITFRAC	80	50	100
NLOG	3	3	3
NUMSP	64	1	256
NUMTIM	16	1	8192
NUMTRW	16	1	8192
* Semaphore Parameters			
SEMAP	10	10	8192
SEMMNI	10	10	8192
SEMMNS	60	60	8192
SEMMNU	30	10	8192
SEMMSL	25	25	150
SEMOPM	10	10	1024
SEMUME	10	10	25
SEMVMX	32767	32767	32767
SEMAEM	16384	16384	16384
* Shared Memory Parameters			
SHMMAX	524288	131072	2147483647
SHMMIN	1	1	1
SHMMNI	100	100	2000
FILE	0	100	64000
NMOUNT	0	4	256
NPROC	0	50	16000
NREGION	0	500	160000

We recommend setting these values as follows:

- `NOFILES` should be 4096 or 2048.
- `MAXUP` should be 2048.

To make changes to the kernel, use the `idtune name parameter` command. `idtune` modifies the `/etc/conf/cf.d/stune` file for you. For example, to change `SEMMS` to `200`, execute this command as `root`:

```
# /etc/conf/bin/idtune SEMMS 200
```

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/idbuild -B && init 6
```

We recommend tuning the system, but the proper parameter values to use depend on the number of users accessing the application or database and size the of the database (that is, the used buffer pool). The following kernel parameters can be set with `idtune`:

- `SHMMAX` (recommended setting: 128MB) and `SHMSEG` (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- `NOFILES` and `MAXUP` should be set to at least 2048.
- `MAXPROC` should be set to at least 3000/4000 (depends on number of users) or more.
- We also recommend using the following formulas to calculate values for `SEMMSL`, `SEMMS`, and `SEMMNU`:

```
SEMMSL = 13
```

13 is what has been found to be the best for both Progress and MySQL.

```
SEMMS = SEMMSL × number of db servers to be run on the system
```

Set `SEMMS` to the value of `SEMMSL` multiplied by the number of database servers (maximum) that you are running on the system at one time.

```
SEMMNU = SEMMS
```

Set the value of `SEMMNU` to equal the value of `SEMMS`. You could probably set this to 75% of `SEMMS`, but this is a conservative estimate.

You need to at least install the SCO OpenServer Linker and Application Development Libraries or the OpenServer Development System to use `gcc`. You cannot use the GCC Dev system without installing one of these.

You should get the FSU Pthreads package and install it first. This can be found at <http://moss.csc.ncsu.edu/~mueller/ftp/pub/PART/pthreads.tar.gz>. You can also get a precompiled package from <ftp://ftp.zenez.com/pub/zenez/prgms/FSU-threads-3.14.tar.gz>.

FSU Pthreads can be compiled with SCO Unix 4.2 with `tcip`, or using OpenServer 3.0 or Open Desktop 3.0 (OS 3.0 ODT 3.0) with the SCO Development System installed using a good port of GCC 2.5.x. For ODT or OS 3.0, you need a good port of GCC 2.5.x. There are a lot of problems without a good port. The port for this product requires the SCO Unix Development system. Without it, you are missing the libraries and the linker that is needed. You also need `SCO-3.2v4.2-includes.tar.gz`. This file contains the changes to the SCO Development include files that are needed to get MySQL to build. You need to replace the existing system include files with these modified header files. They can be obtained from <ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz>.

To build FSU Pthreads on your system, all you should need to do is run GNU `make`. The `Makefile` in `FSU-threads-3.14.tar.gz` is set up to make FSU-threads.

You can run `./configure` in the `threads/src` directory and select the SCO OpenServer option. This command copies `Makefile.SCO5` to `Makefile`. Then run `make`.

To install in the default `/usr/include` directory, log in as `root`, and then `cd` to the `thread/src` directory and run `make install`.

Remember that you must use GNU `make` to build MySQL.

Note

If you don't start `mysqld_safe` as `root`, you should get only the default 110 open files per process. `mysqld` writes a note about this in the log file.

With SCO 3.2V4.2, you should use FSU Pthreads version 3.14 or newer. The following `configure` command should work:

```
CFLAGS="-D_XOPEN_XPG4" CXX=gcc CXXFLAGS="-D_XOPEN_XPG4" \  
./configure \  
  --prefix=/usr/local/mysql \  
  --with-named-thread-libs="-lgthreads -lsocket -lgen -lgthreads" \  
  --with-named-curses-libs="-lcurses"
```

You may have problems with some include files. In this case, you can find new SCO-specific include files at <ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz>.

You should unpack this file in the `include` directory of your MySQL source tree.

SCO development notes:

- MySQL should automatically detect FSU Pthreads and link `mysqld` with `-lgthreads -lsocket -lgthreads`.
- The SCO development libraries are re-entrant in FSU Pthreads. SCO claims that its library functions are re-entrant, so they must be re-entrant with FSU Pthreads. FSU Pthreads on OpenServer tries to use the SCO scheme to make re-entrant libraries.
- FSU Pthreads (at least the version at <ftp://ftp.zenez.com>) comes linked with GNU `malloc`. If you encounter problems with memory usage, make sure that `gmalloc.o` is included in `libgthreads.a` and `libgthreads.so`.
- In FSU Pthreads, the following system calls are pthreads-aware: `read()`, `write()`, `getmsg()`, `connect()`, `accept()`, `select()`, and `wait()`.
- The CSSA-2001-SCO.35.2 (the patch is listed in custom as `erg711905-dscr_remap` security patch (version 2.0.0)) breaks FSU threads and makes `mysqld` unstable. You have to remove this one if you want to run `mysqld` on an OpenServer 5.0.6 machine.
- If you use SCO OpenServer 5, you may need to recompile FSU pthreads with `-DDRAFT7` in `CFLAGS`. Otherwise, `InnoDB` may hang at a `mysqld` startup.
- SCO provides operating system patches at <ftp://ftp.sco.com/pub/openserver5> for OpenServer 5.0.x.
- SCO provides security fixes and `libsocket.so.2` at <ftp://ftp.sco.com/pub/security/OpenServer> and <ftp://ftp.sco.com/pub/security/sse> for OpenServer 5.0.x.
- Pre-OSR506 security fixes. Also, the `telnetd` fix at <ftp://stage.caldera.com/pub/security/openserver/> or <ftp://stage.caldera.com/pub/security/openserver/CSSA-2001-SCO.10/> as both `libsocket.so.2` and `libresolv.so.1` with instructions for installing on pre-OSR506 systems.

It's probably a good idea to install these patches before trying to compile/use MySQL.

Beginning with Legend/OpenServer 6.0.0, there are native threads and no 2GB file size limit.

2.13.5.9. SCO OpenServer 6.0.x Notes

OpenServer 6 includes these key improvements:

- Larger file support up to 1 TB

- Multiprocessor support increased from 4 to 32 processors
- Increased memory support up to 64GB
- Extending the power of UnixWare into OpenServer 6
- Dramatic performance improvement

OpenServer 6.0.0 commands are organized as follows:

- `/bin` is for commands that behave exactly the same as on OpenServer 5.0.x.
- `/u95/bin` is for commands that have better standards conformance, for example Large File System (LFS) support.
- `/udk/bin` is for commands that behave the same as on UnixWare 7.1.4. The default is for the LFS support.

The following is a guide to setting `PATH` on OpenServer 6. If the user wants the traditional OpenServer 5.0.x then `PATH` should be `/bin` first. If the user wants LFS support, the path should be `/u95/bin:/bin`. If the user wants UnixWare 7 support first, the path would be `/udk/bin:/u95/bin:/bin`.

We recommend using the latest production release of MySQL. Should you choose to use an older release of MySQL on OpenServer 6.0.x, you must use a version of MySQL at least as recent as 3.22.13 to get fixes for some portability and OS problems.

MySQL distribution files with names of the following form are `tar` archives of media are tar archives of media images suitable for installation with the SCO Software Manager (`/etc/custom`) on SCO OpenServer 6:

```
mysql-PRODUCT-5.1.21-beta-sco-osr6-i686.VOLS.tar
```

A distribution where `PRODUCT` is `pro-cert` is the Commercially licensed MySQL Pro Certified server. A distribution where `PRODUCT` is `pro-gpl-cert` is the MySQL Pro Certified server licensed under the terms of the General Public License (GPL).

Select whichever distribution you wish to install and, after download, extract the `tar` archive into an empty directory. For example:

```
shell> mkdir /tmp/mysql-pro
shell> cd /tmp/mysql-pro
shell> tar xf /tmp/mysql-pro-cert-5.1.21-beta-sco-osr6-i686.VOLS.tar
```

Prior to installation, back up your data in accordance with the procedures outlined in [Section 2.11, “Upgrading MySQL”](#).

Remove any previously installed `pkgadd` version of MySQL:

```
shell> pkginfo mysql 2>&1 > /dev/null && pkgrm mysql
```

Install MySQL Pro from media images using the SCO Software Manager:

```
shell> /etc/custom -p SCO:MySQL -i -z /tmp/mysql-pro
```

Alternatively, the SCO Software Manager can be displayed graphically by clicking on the [Software Manager](#) icon on the desktop, selecting [Software -> Install New](#), selecting the host, selecting [Media Images](#) for the Media Device, and entering `/tmp/mysql-pro` as the Image Directory.

After installation, run `mkdev mysql` as the `root` user to configure your newly installed MySQL Pro Certified server.

Note

The installation procedure for VOLS packages does not create the `mysql` user and group that the package uses by default. You should either create the `mysql` user and group, or else select a different user and group using an option in `mkdev mysql`.

If you wish to configure your MySQL Pro server to interface with the Apache Web server via PHP, download and install the PHP update from SCO at <http://ftp.sco.com/pub/updates/OpenServer/SCOSA-2006.17/>.

We have been able to compile MySQL with the following `configure` command on OpenServer 6.0.x:

```
CC=cc CFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
CXX=CC CXXFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
./configure --prefix=/usr/local/mysql \
--enable-thread-safe-client \
--with-extra-charsets=complex \
--build=i686-unknown-sysv5SCO_SV6.0.0
```

If you use `gcc`, you must use `gcc` 2.95.3 or newer.

```
CC=gcc CXX=g++ ... ./configure ...
```

SCO provides OpenServer 6 operating system patches at <ftp://ftp.sco.com/pub/openserver6>.

SCO provides information about security fixes at <ftp://ftp.sco.com/pub/security/OpenServer>.

By default, the maximum file size on a OpenServer 6.0.0 system is 1TB. Some operating system utilities have a limitation of 2GB. The maximum possible file size on UnixWare 7 is 1TB with VXFS or HTFS.

OpenServer 6 can be configured for large file support (file sizes greater than 2GB) by tuning the UNIX kernel.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

Value	Default	Min	Max
-----	-----	---	---
SVMMLIM	0x9000000	0x1000000	0x7FFFFFFF
HVMMLIM	0x9000000	0x1000000	0x7FFFFFFF

To make changes to the kernel, use the `idtune name parameter` command. `idtune` modifies the `/etc/conf/cf.d/stune` file for you. We recommend setting the kernel values by executing the following commands as `root`:

```
# /etc/conf/bin/idtune SDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SFNOLIM 2048
# /etc/conf/bin/idtune HFNOLIM 2048
```

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/idbuild -B && init 6
```

We recommend tuning the system, but the proper parameter values to use depend on the number of users accessing the application or database and size of the database (that is, the used buffer pool). The following kernel parameters can be set with `idtune`:

- `SHMMAX` (recommended setting: 128MB) and `SHMSEG` (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- `SFNOLIM` and `HFNOLIM` should be at maximum 2048.
- `NPROC` should be set to at least 3000/4000 (depends on number of users).
- We also recommend using the following formulas to calculate values for `SEMMSL`, `SEMMNS`, and `SEMMNU`:

```
SEMMSL = 13
```

13 is what has been found to be the best for both Progress and MySQL.

```
SEMMNS = SEMMSL × number of db servers to be run on the system
```

Set `SEMMNS` to the value of `SEMMSL` multiplied by the number of database servers (maximum) that you are running on the system at one time.

```
SEMMNU = SEMMNS
```

Set the value of `SEMMNU` to equal the value of `SEMMNS`. You could probably set this to 75% of `SEMMNS`, but this is a conservative estimate.

2.13.5.10. SCO UnixWare 7.1.x and OpenUNIX 8.0.0 Notes

We recommend using the latest production release of MySQL. Should you choose to use an older release of MySQL on UnixWare 7.1.x, you must use a version of MySQL at least as recent as 3.22.13 to get fixes for some portability and OS problems.

We have been able to compile MySQL with the following `configure` command on UnixWare 7.1.x:

```
CC="cc" CFLAGS="-I/usr/local/include" \
CXX="CC" CXXFLAGS="-I/usr/local/include" \
./configure --prefix=/usr/local/mysql \
--enable-thread-safe-client \
--with-innodb --with-openssl --with-extra-charsets=complex
```

If you want to use `gcc`, you must use `gcc 2.95.3` or newer.

```
CC=gcc CXX=g++ ... ./configure ...
```

SCO provides operating system patches at <ftp://ftp.sco.com/pub/unixware7> for UnixWare 7.1.1, <ftp://ftp.sco.com/pub/unixware7/713/> for UnixWare 7.1.3, <ftp://ftp.sco.com/pub/unixware7/714/> for UnixWare 7.1.4, and <ftp://ftp.sco.com/pub/openunix8> for OpenUNIX 8.0.0.

SCO provides information about security fixes at <ftp://ftp.sco.com/pub/security/OpenUNIX> for OpenUNIX and <ftp://ftp.sco.com/pub/security/UnixWare> for UnixWare.

The UnixWare 7 file size limit is 1 TB with VXFS. Some OS utilities have a limitation of 2GB.

On UnixWare 7.1.4 you do not need to do anything to get large file support, but to enable large file support on prior versions of UnixWare 7.1.x, run `fsadm`.

```
# fsadm -Fvxfs -o largefiles /
# fsadm / * Note
# ulimit unlimited
# /etc/conf/bin/ldtune SFSZLIM 0x7FFFFFFF ** Note
# /etc/conf/bin/ldtune HFSZLIM 0x7FFFFFFF ** Note
# /etc/conf/bin/ldbuild -B

* This should report "largefiles".
** 0x7FFFFFFF represents infinity for these values.
```

Reboot the system using `shutdown`.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

Value	Default	Min	Max
SVMMLIM	0x90000000	0x10000000	0x7FFFFFFF
HVMMLIM	0x90000000	0x10000000	0x7FFFFFFF

To make changes to the kernel, use the `ldtune name parameter` command. `ldtune` modifies the `/etc/conf/cf.d/stune` file for you. We recommend setting the kernel values by executing the following commands as `root`:

```
# /etc/conf/bin/ldtune SDATLIM 0x7FFFFFFF
# /etc/conf/bin/ldtune HDATLIM 0x7FFFFFFF
# /etc/conf/bin/ldtune SVMMLIM 0x7FFFFFFF
# /etc/conf/bin/ldtune HVMMLIM 0x7FFFFFFF
# /etc/conf/bin/ldtune SFNOLIM 2048
# /etc/conf/bin/ldtune HFNOLIM 2048
```

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/ldbuild -B && init 6
```

We recommend tuning the system, but the proper parameter values to use depend on the number of users accessing the application or

database and size the of the database (that is, the used buffer pool). The following kernel parameters can be set with `id tune`:

- `SHMMAX` (recommended setting: 128MB) and `SHMSEG` (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- `SFNOLIM` and `HFNOLIM` should be at maximum 2048.
- `NPROC` should be set to at least 3000/4000 (depends on number of users).
- We also recommend using the following formulas to calculate values for `SEMMSL`, `SEMMNS`, and `SEMMNU`:

```
SEMMSL = 13
```

13 is what has been found to be the best for both Progress and MySQL.

```
SEMMNS = SEMMSL × number of db servers to be run on the system
```

Set `SEMMNS` to the value of `SEMMSL` multiplied by the number of database servers (maximum) that you are running on the system at one time.

```
SEMMNU = SEMMNS
```

Set the value of `SEMMNU` to equal the value of `SEMMNS`. You could probably set this to 75% of `SEMMNS`, but this is a conservative estimate.

2.13.6. OS/2 Notes

MySQL uses quite a few open files. Because of this, you should add something like the following to your `CONFIG.SYS` file:

```
SET EMXOPT=-c -n -h1024
```

If you do not do this, you may encounter the following error:

```
File 'xxxx' not found (Errcode: 24)
```

When using MySQL with OS/2 Warp 3, FixPack 29 or above is required. With OS/2 Warp 4, FixPack 4 or above is required. This is a requirement of the Pthreads library. MySQL must be installed on a partition with a type that supports long filenames, such as HPFS, FAT32, and so on.

The `INSTALL.CMD` script must be run from OS/2's own `CMD.EXE` and may not work with replacement shells such as `4OS2.EXE`.

The `scripts/mysql-install-db` script has been renamed. It is called `install.cmd` and is a REXX script, which sets up the default MySQL security settings and creates the WorkPlace Shell icons for MySQL.

Dynamic module support is compiled in but not fully tested. Dynamic modules should be compiled using the Pthreads runtime library.

```
gcc -Zdll -Zmt -Zcrtdll=pthrdrt1 -I../include -I../regex -I. \
-o example udf_example.c -L../lib -lmysqlclient udf_example.def
mv example.dll example.udf
```

Note

Due to limitations in OS/2, UDF module name stems must not exceed eight characters. Modules are stored in the `/mysql2/udf` directory; the `safe-mysqld.cmd` script puts this directory in the `BEGINLIBPATH` environment variable. When using UDF modules, specified extensions are ignored—it is assumed to be `.udf`. For example, in Unix, the shared module might be named `example.so` and you would load a function from it like this:

```
mysql> CREATE FUNCTION metaphor RETURNS STRING SONAME 'example.so';
```

In OS/2, the module would be named `example.udf`, but you would not specify the module extension:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example';
```

2.14. Environment Variables

This section lists all the environment variables that are used directly or indirectly by MySQL. Most of these can also be found in other places in this manual.

Note that any options on the command line take precedence over values specified in option files and environment variables, and values in option files take precedence over values in environment variables.

In many cases, it is preferable to use an option file instead of environment variables to modify the behavior of MySQL. See [Section 4.3.2, “Using Option Files”](#).

Variable	Description
CXX	The name of your C++ compiler (for running configure).
CC	The name of your C compiler (for running configure).
CFLAGS	Flags for your C compiler (for running configure).
CXXFLAGS	Flags for your C++ compiler (for running configure).
DBI_USER	The default username for Perl DBI.
DBI_TRACE	Trace options for Perl DBI.
HOME	The default path for the mysql history file is <code>\$HOME/.mysql_history</code> .
LD_RUN_PATH	Used to specify the location of <code>libmysqlclient.so</code> .
MYSQL_DEBUG	Debug trace options when debugging.
MYSQL_GROUP_SUFFIX	Option group suffix value (like specifying <code>--defaults-group-suffix</code>).
MYSQL_HISTFILE	The path to the mysql history file. If this variable is set, its value overrides the default for <code>\$HOME/.mysql_history</code> .
MYSQL_HOME	The path to the directory in which the server-specific <code>my.cnf</code> file resides (as of MySQL 5.0.3).
MYSQL_HOST	The default hostname used by the mysql command-line client.
MYSQL_PS1	The command prompt to use in the mysql command-line client.
MYSQL_PWD	The default password when connecting to mysqld . Note that using this is insecure. See Section 5.8.6, “Keeping Your Password Secure” .
MYSQL_TCP_PORT	The default TCP/IP port number.
MYSQL_UNIX_PORT	The default Unix socket filename; used for connections to <code>localhost</code> .
PATH	Used by the shell to find MySQL programs.
TMPDIR	The directory where temporary files are created.
TZ	This should be set to your local time zone. See Section B.1.4.6, “Time Zone Problems” .
UMASK_DIR	The user-directory creation mask when creating directories. Note that this is ANDed with <code>UMASK</code> .
UMASK	The user-file creation mask when creating files.
USER	The default username on Windows and NetWare used when connecting to mysqld .

2.15. Perl Installation Notes

Perl support for MySQL is provided by means of the [DBI/DBD](#) client interface. The interface requires Perl 5.6.1 or later. It *does not work* if you have an older version of Perl.

If you want to use transactions with Perl DBI, you need to have `DBD: :mysql` version 1.2216 or newer. `DBD: :mysql` 2.9003 or newer is recommended.

If you are using the MySQL 4.1 or newer client library, you must use `DBD: :mysql` 2.9003 or newer.

Perl support is not included with MySQL distributions. You can obtain the necessary modules from <http://search.cpan.org> for Unix, or

by using the ActiveState [ppm](#) program on Windows. The following sections describe how to do this.

Perl support for MySQL must be installed if you want to run the MySQL benchmark scripts. See [Section 6.1.4](#), “The MySQL Benchmark Suite”.

2.15.1. Installing Perl on Unix

MySQL Perl support requires that you have installed MySQL client programming support (libraries and header files). Most installation methods install the necessary files. However, if you installed MySQL from RPM files on Linux, be sure that you’ve installed the developer RPM. The client programs are in the client RPM, but client programming support is in the developer RPM.

If you want to install Perl support, the files you need can be obtained from the CPAN (Comprehensive Perl Archive Network) at <http://search.cpan.org>.

The easiest way to install Perl modules on Unix is to use the [CPAN](#) module. For example:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD:mysql
```

The `DBD:mysql` installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default username and password. (The default username is your login name on Unix, and `ODBC` on Windows. The default password is “no password.”) If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use `force install DBD:mysql` to ignore the failed tests.

`DBI` requires the `Data:Dumper` module. It may be installed; if not, you should install it before installing `DBI`.

It is also possible to download the module distributions in the form of compressed `tar` archives and build the modules manually. For example, to unpack and build a `DBI` distribution, use a procedure such as this:

1. Unpack the distribution into the current directory:

```
shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
```

This command creates a directory named `DBI-VERSION`.

2. Change location into the top-level directory of the unpacked distribution:

```
shell> cd DBI-VERSION
```

3. Build the distribution and compile everything:

```
shell> perl Makefile.PL
shell> make
shell> make test
shell> make install
```

The `make test` command is important because it verifies that the module is working. Note that when you run that command during the `DBD:mysql` installation to exercise the interface code, the MySQL server must be running or the test fails.

It is a good idea to rebuild and reinstall the `DBD:mysql` distribution whenever you install a new release of MySQL, particularly if you notice symptoms such as that all your `DBI` scripts fail after you upgrade MySQL.

If you do not have access rights to install Perl modules in the system directory or if you want to install local Perl modules, the following reference may be useful: <http://servers.digitaldaze.com/extensions/perl/modules.html#modules>

Look under the heading “Installing New Modules that Require Locally Installed Modules.”

2.15.2. Installing ActiveState Perl on Windows

On Windows, you should do the following to install the MySQL `DBD` module with ActiveState Perl:

1. Get ActiveState Perl from <http://www.activestate.com/Products/ActivePerl/> and install it.

2. Open a console window (a “DOS window”).
3. If necessary, set the `HTTP_proxy` variable. For example, you might try a setting like this:

```
set HTTP_proxy=my.proxy.com:3128
```

4. Start the PPM program:

```
C:\> C:\perl\bin\ppm.pl
```

5. If you have not previously done so, install `DBI`:

```
ppm> install DBI
```

6. If this succeeds, run the following command:

```
ppm> install DBD-mysql
```

This procedure should work with ActiveState Perl 5.6 or newer.

If you cannot get the procedure to work, you should install the MyODBC driver instead and connect to the MySQL server through ODBC:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
    die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.15.3. Problems Using the Perl `DBI/DBD` Interface

If Perl reports that it cannot find the `../mysql/mysql.so` module, the problem is probably that Perl cannot locate the `libmysqlclient.so` shared library. You should be able to fix this problem by one of the following methods:

- Compile the `DBD:mysql` distribution with `perl Makefile.PL -static -config` rather than `perl Makefile.PL`.
- Copy `libmysqlclient.so` to the directory where your other shared libraries are located (probably `/usr/lib` or `/lib`).
- Modify the `-L` options used to compile `DBD:mysql` to reflect the actual location of `libmysqlclient.so`.
- On Linux, you can add the pathname of the directory where `libmysqlclient.so` is located to the `/etc/ld.so.conf` file.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable. Some systems use `LD_LIBRARY_PATH` instead.

Note that you may also need to modify the `-L` options if there are other libraries that the linker fails to find. For example, if the linker cannot find `libc` because it is in `/lib` and the link command specifies `-L/usr/lib`, change the `-L` option to `-L/lib` or add `-L/lib` to the existing link command.

If you get the following errors from `DBD:mysql`, you are probably using `gcc` (or using an old binary compiled with `gcc`):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add `-L/usr/lib/gcc-lib/... -lgcc` to the link command when the `mysql.so` library gets built (check the output from `make` for `mysql.so` when you compile the Perl client). The `-L` option should specify the pathname of the directory where `libgcc.a` is located on your system.

Another cause of this problem may be that Perl and MySQL are not both compiled with `gcc`. In this case, you can solve the mismatch by compiling both with `gcc`.

You may see the following error from `DBD:mysql` when you run the tests:


```
t/00base.....install_driver(mysql) failed:
Can't load './b/lib/arch/auto/DBD/mysql/mysql.so' for module DBD::mysql:
../b/lib/arch/auto/DBD/mysql/mysql.so: undefined symbol:
uncompress at /usr/lib/perl5/5.00503/i586-linux/DynaLoader.pm line 169.
```

This means that you need to include the `-lz` compression library on the link line. That can be done by changing the following line in the file `lib/DBD/mysql/Install.pm`:

```
$sysliblist .= " -lm";
```

Change that line to:

```
$sysliblist .= " -lm -lz";
```

After this, you *must* run `make realclean` and then proceed with the installation from the beginning.

If you want to install DBI on SCO, you have to edit the `Makefile` in `DBI-xxx` and each subdirectory. Note that the following assumes `gcc 2.95.2` or newer:

OLD:	NEW:
CC = cc	CC = gcc
CCCDLFLAGS = -KPIC -Wl,-Bexport	CCCDLFLAGS = -fpic
CCDLFLAGS = -wl,-Bexport	CCDLFLAGS =
LD = ld	LD = gcc -G -fpic
LDDLFLAGS = -G -L/usr/local/lib	LDDLFLAGS = -L/usr/local/lib
LDFLAGS = -belf -L/usr/local/lib	LDFLAGS = -L/usr/local/lib
LD = ld	LD = gcc -G -fpic
OPTIMIZE = -Od	OPTIMIZE = -O1
OLD:	
CCCFLAGS = -belf -dy -w0 -U M_XENIX -DPERL_SC05 -I/usr/local/include	
NEW:	
CCFLAGS = -U M_XENIX -DPERL_SC05 -I/usr/local/include	

These changes are necessary because the Perl dynaloader does not load the `DBI` modules if they were compiled with `icc` or `cc`.

If you want to use the Perl module on a system that does not support dynamic linking (such as SCO), you can generate a static version of Perl that includes `DBI` and `DBD::mysql`. The way this works is that you generate a version of Perl with the `DBI` code linked in and install it on top of your current Perl. Then you use that to build a version of Perl that additionally has the `DBD` code linked in, and install that.

On SCO, you must have the following environment variables set:

```
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
```

Or:

```
LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
/usr/progressive/lib:/usr/skunk/lib
LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
/usr/progressive/lib:/usr/skunk/lib
MANPATH=scohelp:/usr/man:/usr/local1/man:/usr/local/man:\
/usr/skunk/man:
```

First, create a Perl that includes a statically linked `DBI` module by running these commands in the directory where your `DBI` distribution is located:

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

Then you must install the new Perl. The output of `make perl` indicates the exact `make` command you need to execute to perform the installation. On SCO, this is `make -f Makefile.aperl inst_perl MAP_TARGET=perl`.

Next, use the just-created Perl to create another Perl that also includes a statically linked `DBD::mysql` by running these commands in the directory where your `DBD::mysql` distribution is located:

```

shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl

```

Finally, you should install this new Perl. Again, the output of `make perl` indicates the command to use.

2.16. Porting to Other Systems

This appendix helps you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See [Section 2.1.1, “Operating Systems Supported by MySQL Community Server”](#). If you have created a new port of MySQL, please let us know so that we can list it here and on our Web site (<http://www.mysql.com/>), recommending it to other users.

Note: If you create a new port of MySQL, you are free to copy and distribute it under the GPL license, but it does not make you a copy-right holder of MySQL.

A working POSIX thread library is needed for the server. On Solaris 2.5 we use Sun PThreads (the native thread support in 2.4 and earlier versions is not good enough), on Linux we use LinuxThreads by Xavier Leroy, <Xavier.Leroy@inria.fr>.

The hard part of porting to a new Unix variant without good native thread support is probably to port MIT-pthreads. See [mit-pthreads/README](#) and Programming POSIX Threads (<http://www.humanfactor.com/pthreads/>).

Up to MySQL 4.0.2, the MySQL distribution included a patched version of Chris Provenzano's Pthreads from MIT (see the MIT Pthreads Web page at <http://www.mit.edu/afs/sipb/project/pthreads/> and a programming introduction at http://www.mit.edu:8001/people/proven/IAP_2000/). These can be used for some operating systems that do not have POSIX threads. See [Section 2.9.5, “MIT-pthreads Notes”](#).

It is also possible to use another user level thread package named FSU Pthreads (see <http://moss.csc.ncsu.edu/~mueller/pthreads/>). This implementation is being used for the SCO port.

See the `thr_lock.c` and `thr_alarm.c` programs in the `mysys` directory for some tests/examples of these problems.

Both the server and the client need a working C++ compiler. We use `gcc` on many platforms. Other compilers that are known to work are SPARCworks, Sun Forte, Irix `cc`, HP-UX `aCC`, IBM AIX `xlc_r`, Intel `ecc/icc` and Compaq `cxx`.

Important

If you are trying to build MySQL 5.1 with `icc` on the IA64 platform, and need support for MySQL Cluster, you should first ensure that you are using `icc` version 9.1.043 or later. (For details, see [Bug#21875](#).)

To compile only the client use `./configure --without-server`.

There is currently no support for only compiling the server, nor is it likely to be added unless someone has a good reason for it.

If you want/need to change any `Makefile` or the configure script you also need GNU Automake and Autoconf. See [Section 2.9.3, “Installing from the Development Source Tree”](#).

All steps needed to remake everything from the most basic files.

```

/bin/rm */.deps/*.P
/bin/rm -f config.cache
aclocal
autoheader
aclocal
automake
autoconf
./configure --with-debug=full --prefix='your installation directory'

# The makefiles generated above need GNU make 3.75 or newer.
# (called gmake below)
gmake clean all install init-db

```

If you run into problems with a new port, you may have to do some debugging of MySQL! See [Section 2.16.1, “Debugging a MySQL Server”](#).

Note

Before you start debugging `mysqld`, first get the test programs `mysys/thr_alarm` and `mysys/thr_lock` to work.

■ This ensures that your thread installation has even a remote chance to work!

2.16.1. Debugging a MySQL Server

If you are using some functionality that is very new in MySQL, you can try to run `mysqld` with the `--skip-new` (which disables all new, potentially unsafe functionality) or with `--safe-mode` which disables a lot of optimization that may cause problems. See [Section B.1.4.2, “What to Do If MySQL Keeps Crashing”](#).

If `mysqld` doesn't want to start, you should verify that you don't have any `my.cnf` files that interfere with your setup! You can check your `my.cnf` arguments with `mysqld --print-defaults` and avoid using them by starting with `mysqld --no-defaults`

If `mysqld` starts to eat up CPU or memory or if it “hangs,” you can use `mysqladmin processlist status` to find out if someone is executing a query that takes a long time. It may be a good idea to run `mysqladmin -i10 processlist status` in some window if you are experiencing performance problems or problems when new clients can't connect.

The command `mysqladmin debug` dumps some information about locks in use, used memory and query usage to the MySQL log file. This may help solve some problems. This command also provides some useful information even if you haven't compiled MySQL for debugging!

If the problem is that some tables are getting slower and slower you should try to optimize the table with `OPTIMIZE TABLE` or `my-isamchk`. See [Chapter 5, Database Administration](#). You should also check the slow queries with `EXPLAIN`.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See [Section 2.13, “Operating System-Specific Notes”](#).

2.16.1.1. Compiling MySQL for Debugging

If you have some very specific problem, you can always try to debug MySQL. To do this you must configure MySQL with the `--with-debug` or the `--with-debug=full` option. You can check whether MySQL was compiled with debugging by doing: `mysqld --help`. If the `--debug` flag is listed with the options then you have debugging enabled. `mysqladmin ver` also lists the `mysqld` version as `mysql . . . --debug` in this case.

If you are using `gcc` or `egcs`, the recommended `configure` line is:

```
CC=gcc CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors \
-fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql \
--with-debug --with-extra-charsets=complex
```

This avoids problems with the `libstdc++` library and with C++ exceptions (many compilers have problems with C++ exceptions in threaded code) and compile a MySQL version with support for all character sets.

If you suspect a memory overrun error, you can configure MySQL with `--with-debug=full`, which installs a memory allocation (`SAFEMALLOC`) checker. However, running with `SAFEMALLOC` is quite slow, so if you get performance problems you should start `mysqld` with the `--skip-safemalloc` option. This disables the memory overrun checks for each call to `malloc()` and `free()`.

If `mysqld` stops crashing when you compile it with `--with-debug`, you probably have found a compiler bug or a timing bug within MySQL. In this case, you can try to add `-g` to the `CFLAGS` and `CXXFLAGS` variables above and not use `--with-debug`. If `mysqld` dies, you can at least attach to it with `gdb` or use `gdb` on the core file to find out what happened.

When you configure MySQL for debugging you automatically enable a lot of extra safety check functions that monitor the health of `mysqld`. If they find something “unexpected,” an entry is written to `stderr`, which `mysqld_safe` directs to the error log! This also means that if you are having some unexpected problems with MySQL and are using a source distribution, the first thing you should do is to configure MySQL for debugging! (The second thing is to send mail to a MySQL mailing list and ask for help. See [Section 1.7.1, “MySQL Mailing Lists”](#). If you believe that you have found a bug, please use the instructions at [Section 1.8, “How to Report Bugs or Problems”](#)).

In the Windows MySQL distribution, `mysqld.exe` is by default compiled with support for trace files.

2.16.1.2. Creating Trace Files

If the `mysqld` server doesn't start or if you can cause it to crash quickly, you can try to create a trace file to find the problem.

To do this, you must have a `mysqld` that has been compiled with debugging support. You can check this by executing `mysqld -V`. If the version number ends with `-debug`, it's compiled with support for trace files. (On Windows, the debugging server is named

`mysqld-debug` rather than `mysqld` as of MySQL 4.1.)

Start the `mysqld` server with a trace log in `/tmp/mysqld.trace` on Unix or `C:\mysqld.trace` on Windows:

```
shell> mysqld --debug
```

On Windows, you should also use the `--standalone` flag to not start `mysqld` as a service. In a console window, use this command:

```
C:\> mysqld-debug --debug --standalone
```

After this, you can use the `mysql.exe` command-line tool in a second console window to reproduce the problem. You can stop the `mysqld` server with `mysqladmin shutdown`.

Note that the trace file become **very big**! If you want to generate a smaller trace file, you can use debugging options something like this:

```
mysqld --debug=d,info,error,query,general,where:0,/tmp/mysqld.trace
```

This only prints information with the most interesting tags to the trace file.

If you make a bug report about this, please only send the lines from the trace file to the appropriate mailing list where something seems to go wrong! If you can't locate the wrong place, you can ftp the trace file, together with a full bug report, to <ftp://ftp.mysql.com/pub/mysql/upload/> so that a MySQL developer can take a look at it.

The trace file is made with the **DEBUG** package by Fred Fish. See [Section 2.16.3, “The DEBUG Package”](#).

2.16.1.3. Debugging `mysqld` under `gdb`

On most systems you can also start `mysqld` from `gdb` to get more information if `mysqld` crashes.

With some older `gdb` versions on Linux you must use `run --one-thread` if you want to be able to debug `mysqld` threads. In this case, you can only have one thread active at a time. We recommend you to upgrade to `gdb` 5.1 ASAP as thread debugging works much better with this version!

NPTL threads (the new thread library on Linux) may cause problems while running `mysqld` under `gdb`. Some symptoms are:

- `mysqld` hangs during startup (before it writes `ready for connections`).
- `mysqld` crashes during a `pthread_mutex_lock()` or `pthread_mutex_unlock()` call.

In this case, you should set the following environment variable in the shell before starting `gdb`:

```
LD_ASSUME_KERNEL=2.4.1  
export LD_ASSUME_KERNEL
```

When running `mysqld` under `gdb`, you should disable the stack trace with `--skip-stack-trace` to be able to catch segfaults within `gdb`.

In MySQL 4.0.14 and above you should use the `--gdb` option to `mysqld`. This installs an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling.

It's very hard to debug MySQL under `gdb` if you do a lot of new connections the whole time as `gdb` doesn't free the memory for old threads. You can avoid this problem by starting `mysqld` with `--thread_cache_size='max_connections+1'`. In most cases just using `--thread_cache_size=5` helps a lot!

If you want to get a core dump on Linux if `mysqld` dies with a `SIGSEGV` signal, you can start `mysqld` with the `--core-file` option. This core file can be used to make a backtrace that may help you find out why `mysqld` died:

```
shell> gdb mysqld core  
gdb> backtrace full  
gdb> quit
```

See [Section B.1.4.2, “What to Do If MySQL Keeps Crashing”](#).

If you are using `gdb` 4.17.x or above on Linux, you should install a `.gdb` file, with the following information, in your current direct-

ory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

If you have problems debugging threads with [gdb](#), you should download [gdb 5.x](#) and try this instead. The new [gdb](#) version has very improved thread handling!

Here is an example how to debug `mysqld`:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

Include the above output in a bug report, which you can file using the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

If `mysqld` hangs you can try to use some system tools like [strace](#) or `/usr/proc/bin/pstack` to examine where `mysqld` has hung.

```
strace /tmp/log libexec/mysqld
```

If you are using the Perl [DBI](#) interface, you can turn on debugging information by using the [trace](#) method or by setting the [DBI_TRACE](#) environment variable.

2.16.1.4. Using a Stack Trace

On some operating systems, the error log contains a stack trace if `mysqld` dies unexpectedly. You can use this to find out where (and maybe why) `mysqld` died. See [Section 5.11.2, “The Error Log”](#). To get a stack trace, you must not compile `mysqld` with the [-fomit-frame-pointer](#) option to gcc. See [Section 2.16.1.1, “Compiling MySQL for Debugging”](#).

If the error file contains something like the following:

```
mysqld got signal 11;
The manual section 'Debugging a MySQL server' tells you how to
use a stack trace and/or the core file to produce a readable
backtrace that may help in finding out why mysqld died
Attempting backtrace. You can use the following information
to find out where mysqld died. If you see no messages after
this, something went terribly wrong...
stack range sanity check, ok, backtrace follows
0x40077552
0x81281a0
0x8128f47
0x8127be0
0x8127995
0x8104947
0x80ff28f
0x810131b
0x80ee4bc
0x80c3c91
0x80c6b43
0x80c1fd9
0x80c1686
```

you can find where `mysqld` died by doing the following:

1. Copy the preceding numbers to a file, for example `mysqld.stack`.
2. Make a symbol file for the `mysqld` server:

```
nm -n libexec/mysqld > /tmp/mysqld.sym
```

Note that most MySQL binary distributions (except for the “debug” packages, where this information is included inside of the binaries themselves) ship with the above file, named `mysqld.sym.gz`. In this case, you can simply unpack it by doing:

```
gunzip < bin/mysqld.sym.gz > /tmp/mysqld.sym
```

3. Execute `resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack`.

This prints out where `mysqld` died. If this doesn't help you find out why `mysqld` died, you should make a bug report and include the output from the above command with the bug report.

Note however that in most cases it does not help us to just have a stack trace to find the reason for the problem. To be able to locate the bug or provide a workaround, we would in most cases need to know the query that killed `mysqld` and preferable a test case so that we can repeat the problem! See [Section 1.8, “How to Report Bugs or Problems”](#).

2.16.1.5. Using Server Logs to Find Causes of Errors in `mysqld`

Note that before starting `mysqld` with `--log` you should check all your tables with `myisamchk`. See [Chapter 5, Database Administration](#).

If `mysqld` dies or hangs, you should start `mysqld` with `--log`. When `mysqld` dies again, you can examine the end of the log file for the query that killed `mysqld`.

If you are using `--log` without a file name, the log is stored in the database directory as `host_name.log`. In most cases it is the last query in the log file that killed `mysqld`, but if possible you should verify this by restarting `mysqld` and executing the found query from the `mysql` command-line tools. If this works, you should also test all complicated queries that didn't complete.

You can also try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that `mysqld` is using indexes properly. See [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#).

You can find the queries that take a long time to execute by starting `mysqld` with `--log-slow-queries`. See [Section 5.11.5, “The Slow Query Log”](#).

If you find the text `mysqld restarted` in the error log file (normally named `hostname.err`) you probably have found a query that causes `mysqld` to fail. If this happens, you should check all your tables with `myisamchk` (see [Chapter 5, Database Administration](#)), and test the queries in the MySQL log files to see whether one fails. If you find such a query, try first upgrading to the newest MySQL version. If this doesn't help and you can't find anything in the `mysql` mail archive, you should report the bug to a MySQL mailing list. The mailing lists are described at <http://lists.mysql.com/>, which also has links to online list archives.

If you have started `mysqld` with `myisam-recover`, MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file 'Warning: Checking table ...' which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further. See [Section 5.2.2, “Command Options”](#).

It is not a good sign if `mysqld` did die unexpectedly, but in this case, you should not investigate the `Checking table...` messages, but instead try to find out why `mysqld` died.

2.16.1.6. Making a Test Case If You Experience Table Corruption

If you get corrupted tables or if `mysqld` always fails after some update commands, you can test whether this bug is reproducible by doing the following:

- Take down the MySQL daemon (with `mysqldadmin shutdown`).
- Make a backup of the tables (to guard against the very unlikely case that the repair does something bad).
- Check all tables with `myisamchk -s database/*.MYI`. Repair any wrong tables with `myisamchk -r database/table.MYI`.
- Make a second backup of the tables.
- Remove (or move away) any old log files from the MySQL data directory if you need more space.
- Start `mysqld` with `--log-bin`. See [Section 5.11.4, “The Binary Log”](#). If you want to find a query that crashes `mysqld`, you should use `--log --log-bin`.

- When you have gotten a crashed table, stop the `mysqld` server.
- Restore the backup.
- Restart the `mysqld` server **without** `--log-bin`
- Re-execute the commands with `mysqlbinlog binary-log-file | mysql`. The binary log is saved in the MySQL data-base directory with the name `hostname-bin.#`.
- If the tables are corrupted again or you can get `mysqld` to die with the above command, you have found reproducible bug that should be easy to fix! FTP the tables and the binary log to <ftp://ftp.mysql.com/pub/mysql/upload/> and report it in our bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#). (Please note that the `/pub/mysql/upload/` FTP directory is not listable, so you'll not see what you've uploaded in your FTP client.) If you are a support customer, you can use the MySQL Customer Support Center <https://support.mysql.com/> to alert the MySQL team about the problem and have it fixed as soon as possible.

You can also use the script `mysql_find_rows` to just execute some of the update statements if you want to narrow down the problem.

2.16.2. Debugging a MySQL Client

To be able to debug a MySQL client with the integrated debug package, you should configure MySQL with `--with-debug` or `--with-debug=full`. See [Section 2.9.2, “Typical configure Options”](#).

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:0,/tmp/client.trace
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in `/tmp/client.trace`.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running `mysql` in debugging mode (assuming that you have compiled MySQL with debugging on):

```
shell> mysql --debug=d:t:0,/tmp/client.trace
```

This provides useful information in case you mail a bug report. See [Section 1.8, “How to Report Bugs or Problems”](#).

If your client crashes at some 'legal' looking code, you should check that your `mysql.h` include file matches your MySQL library file. A very common mistake is to use an old `mysql.h` file from an old MySQL installation with new MySQL library.

2.16.3. The DBUG Package

The MySQL server and most MySQL clients are compiled with the DBUG package originally created by Fred Fish. When you have configured MySQL for debugging, this package makes it possible to get a trace file of what the program is debugging. See [Section 2.16.1.2, “Creating Trace Files”](#).

This section summarizes the argument values that you can specify in debug options on the command line for MySQL programs that have been built with debugging support. For more information about programming with the DBUG package, see the DBUG manual in the `dbug` directory of MySQL source distributions. It's best to use a recent distribution to get the most updated DBUG manual.

You use the debug package by invoking a program with the `--debug="..."` or the `-#...` option.

Most MySQL programs have a default debug string that is used if you don't specify an option to `--debug`. The default trace file is usually `/tmp/program_name.trace` on Unix and `\program_name.trace` on Windows.

The debug control string is a sequence of colon-separated fields as follows:

```
<field_1>:<field_2>:...:<field_N>
```

Each field consists of a mandatory flag character followed by an optional `' , '` and comma-separated list of modifiers:

```
flag[,modifier,modifier,...,modifier]
```


The currently recognized flag characters are:

Flag	Description
d	Enable output from <code>DEBUG_<N></code> macros for the current state. May be followed by a list of keywords which selects output only for the <code>DEBUG</code> macros with that keyword. An empty list of keywords implies output for all macros.
D	Delay after each debugger output line. The argument is the number of tenths of seconds to delay, subject to machine capabilities. For example, <code>-#D,20</code> specifies a delay of two seconds.
f	Limit debugging, tracing, and profiling to the list of named functions. Note that a null list disables all functions. The appropriate d or t flags must still be given; this flag only limits their actions if they are enabled.
F	Identify the source file name for each line of debug or trace output.
i	Identify the process with the PID or thread ID for each line of debug or trace output.
g	Enable profiling. Create a file called <code>debugmon.out</code> containing information that can be used to profile the program. May be followed by a list of keywords that select profiling only for the functions in that list. A null list implies that all functions are considered.
L	Identify the source file line number for each line of debug or trace output.
n	Print the current function nesting depth for each line of debug or trace output.
N	Number each line of debug output.
o	Redirect the debugger output stream to the specified file. The default output is <code>stderr</code> .
O	Like o , but the file is really flushed between each write. When needed, the file is closed and reopened between each write.
p	Limit debugger actions to specified processes. A process must be identified with the <code>DEBUG_PROCESS</code> macro and match one in the list for debugger actions to occur.
P	Print the current process name for each line of debug or trace output.
r	When pushing a new state, do not inherit the previous state's function nesting level. Useful when the output is to start at the left margin.
S	Do function <code>_sanity(_file_, _line_)</code> at each debugged function until <code>_sanity()</code> returns something that differs from 0. (Mostly used with <code>safemalloc</code> to find memory leaks)
t	Enable function call/exit trace lines. May be followed by a list (containing only one modifier) giving a numeric maximum trace level, beyond which no output occurs for either debugging or tracing macros. The default is a compile time option.

Some examples of debug control strings that might appear on a shell command line (the `-#` is typically used to introduce a control string to an application program) are:

```
-#d:t
-#d:f,main,subr1:F:L:t,20
-#d,input,output,files:n
-#d:t:i:0,\\mysqld.trace
```

In MySQL, common tags to print (with the **d** option) are `enter`, `exit`, `error`, `warning`, `info`, and `loop`.

2.16.4. Comments about RTS Threads

I have tried to use the RTS thread packages with MySQL but stumbled on the following problems:

They use old versions of many POSIX calls and it is very tedious to make wrappers for all functions. I am inclined to think that it would be easier to change the thread libraries to the newest POSIX specification.

Some wrappers are currently written. See [mysys/my_pthread.c](#) for more info.

At least the following should be changed:

`pthread_get_specific` should use one argument. `sigwait` should take two arguments. A lot of functions (at least `pthread_cond_wait`, `pthread_cond_timedwait()`) should return the error code on error. Now they return -1 and set `errno`.

Another problem is that user-level threads use the `ALRM` signal and this aborts a lot of functions (`read`, `write`, `open`...). MySQL should do a retry on interrupt on all of these but it is not that easy to verify it.

The biggest unsolved problem is the following:

To get thread-level alarms I changed `mysys/thr_alarm.c` to wait between alarms with `pthread_cond_timedwait()`, but this aborts with error `EINTR`. I tried to debug the thread library as to why this happens, but couldn't find any easy solution.

If someone wants to try MySQL with RTS threads I suggest the following:

- Change functions MySQL uses from the thread library to POSIX. This shouldn't take that long.
- Compile all libraries with the `-DHAVE_rts_threads`.
- Compile `thr_alarm`.
- If there are some small differences in the implementation, they may be fixed by changing `my_pthread.h` and `my_pthread.c`.
- Run `thr_alarm`. If it runs without any “warning,” “error,” or aborted messages, you are on the right track. Here is a successful run on Solaris:

```
Main thread: 1
Thread 0 (5) started
Thread: 5 Waiting
process_alarm
Thread 1 (6) started
Thread: 6 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 1 (1) sec
Thread: 6 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 2 (2) sec
Thread: 6 Simulation of no alarm needed
Thread: 6 Slept for 0 (3) sec
Thread: 6 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 4 (4) sec
Thread: 6 Waiting
process_alarm
thread_alarm
Thread: 5 Slept for 10 (10) sec
Thread: 5 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 5 (5) sec
Thread: 6 Waiting
process_alarm
process_alarm
...
thread_alarm
Thread: 5 Slept for 0 (1) sec
end
```

2.16.5. Differences Between Thread Packages

MySQL is very dependent on the thread package used. So when choosing a good platform for MySQL, the thread package is very important.

There are at least three types of thread packages:

- User threads in a single process. Thread switching is managed with alarms and the threads library manages all non-thread-safe functions with locks. Read, write and select operations are usually managed with a thread-specific select that switches to another thread if the running threads have to wait for data. If the user thread packages are integrated in the standard libs (FreeBSD and BSDI threads) the thread package requires less overhead than thread packages that have to map all unsafe calls (MIT-pthreads, FSU Pthreads and RTS threads). In some environments (for example, SCO), all system calls are thread-safe so the mapping can be done very easily (FSU Pthreads on SCO). Downside: All mapped calls take a little time and it's quite tricky to be able to handle all situations. There are usually also some system calls that are not handled by the thread package (like MIT-pthreads and sockets). Thread

scheduling isn't always optimal.

- User threads in separate processes. Thread switching is done by the kernel and all data are shared between threads. The thread package manages the standard thread calls to allow sharing data between threads. LinuxThreads is using this method. Downside: Lots of processes. Thread creating is slow. If one thread dies the rest are usually left hanging and you must kill them all before restarting. Thread switching is somewhat expensive.
- Kernel threads. Thread switching is handled by the thread library or the kernel and is very fast. Everything is done in one process, but on some systems, `ps` may show the different threads. If one thread aborts, the whole process aborts. Most system calls are thread-safe and should require very little overhead. Solaris, HP-UX, AIX and OSF/1 have kernel threads.

In some systems kernel threads are managed by integrating user level threads in the system libraries. In such cases, the thread switching can only be done by the thread library and the kernel isn't really “thread aware.”

Chapter 3. Tutorial

This chapter provides a tutorial introduction to MySQL by showing how to use the `mysql` client program to create and use a simple database. `mysql` (sometimes referred to as the “terminal monitor” or just “monitor”) is an interactive program that allows you to connect to a MySQL server, run queries, and view the results. `mysql` may also be used in batch mode: you place your queries in a file beforehand, then tell `mysql` to execute the contents of the file. Both ways of using `mysql` are covered here.

To see a list of options provided by `mysql`, invoke it with the `--help` option:

```
shell> mysql --help
```

This chapter assumes that `mysql` is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If *you* are the administrator, you need to consult the relevant portions of this manual, such as [Chapter 5, Database Administration](#).)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily omitted. Consult the relevant sections of the manual for more information on the topics covered here.

3.1. Connecting to and Disconnecting from the Server

To connect to the server, you will usually need to provide a MySQL user name when you invoke `mysql` and, most likely, a password. If the server runs on a machine other than the one where you log in, you will also need to specify a host name. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: *****
```

`host` and `user` represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The `*****` represents your password; enter it when `mysql` displays the `Enter password:` prompt.

If that works, you should see some introductory information followed by a `mysql>` prompt:

```
shell> mysql -h host -u user -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 5.1.21-beta-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The `mysql>` prompt tells you that `mysql` is ready for you to enter commands.

If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
shell< mysql -u user -p
```

If, when you attempt to log in, you get an error message such as `ERROR 2002 (HY000): CAN'T CONNECT TO LOCAL MYSQL SERVER THROUGH SOCKET '/TMP/MYSQL.SOCK' (2)`, it means that that MySQL server daemon (Unix) or service (Windows) is not running. Consult the administrator or see the section of [Chapter 2, Installing and Upgrading MySQL](#) that is appropriate to your operating system.

For help with other problems often encountered when trying to log in, see [Section B.1.2, “Common Errors When Using MySQL Programs”](#).

Some MySQL installations allow users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking `mysql` without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing **QUIT** (or **\q**) at the **mysql>** prompt:

```
mysql> QUIT
Bye
```

On Unix, you can also disconnect by pressing Control-D.

Most examples in the following sections assume that you are connected to the server. They indicate this by the **mysql>** prompt.

3.2. Entering Queries

Make sure that you are connected to the server, as discussed in the previous section. Doing so does not in itself select any database to work with, but that's okay. At this point, it's more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how **mysql** works.

Here's a simple command that asks the server to tell you its version number and the current date. Type it in as shown here following the **mysql>** prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----+-----+
| VERSION() | CURRENT_DATE |
+-----+-----+
| 5.1.2-alpha-log | 2005-10-11 |
+-----+-----+
1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about **mysql**:

- A command normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. **QUIT**, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a command, **mysql** sends it to the server for execution and displays the results, then prints another **mysql>** prompt to indicate that it is ready for another command.
- **mysql** displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), **mysql** labels the column using the expression itself.
- **mysql** shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the "rows in set" line is sometimes not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), cUrReNt_DaTe;
```

Here's another query. It demonstrates that you can use **mysql** as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-----+-----+
| SIN(PI()/4) | (4+1)*5 |
+-----+-----+
| 0.70710678118655 | 25 |
+-----+-----+
1 row in set (0.02 sec)
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
```

```

+-----+
| VERSION() |
+-----+
| 5.1.2-alpha-log |
+-----+
1 row in set (0.00 sec)

+-----+
| NOW() |
+-----+
| 2005-10-11 15:15:00 |
+-----+
1 row in set (0.00 sec)

```

A command need not be given all on a single line, so lengthy commands that require several lines are not a problem. `mysql` determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, `mysql` accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here's a simple multiple-line statement:

```

mysql> SELECT
-> USER()
->
-> CURRENT_DATE;
+-----+-----+
| USER() | CURRENT_DATE |
+-----+-----+
| jon@localhost | 2005-10-11 |
+-----+-----+

```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how `mysql` indicates that it has not yet seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you can always be aware of what `mysql` is waiting for.

If you decide you do not want to execute a command that you are in the process of entering, cancel it by typing `\c`:

```

mysql> SELECT
-> USER()
-> \c
mysql>

```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that `mysql` is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that `mysql` is in:

Prompt	Meaning
<code>mysql></code>	Ready for new command.
<code>-></code>	Waiting for next line of multiple-line command.
<code>'></code>	Waiting for next line, waiting for completion of a string that began with a single quote ('').
<code>"></code>	Waiting for next line, waiting for completion of a string that began with a double quote ("").
<code>`></code>	Waiting for next line, waiting for completion of an identifier that began with a backtick (`).
<code>/*></code>	Waiting for next line, waiting for completion of a comment that began with /*.

Multiple-line statements commonly occur by accident when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, `mysql` waits for more input:

```

mysql> SELECT USER()
->

```

If this happens to you (you think you've entered a statement but the only response is a `->` prompt), most likely `mysql` is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and `mysql` executes it:

```

mysql> SELECT USER()
-> ;
+-----+
| USER() |
+-----+

```

```
+-----+
| jon@localhost |
+-----+
```

The `'>` and `">` prompts occur during string collection (another way of saying that MySQL is waiting for completion of a string). In MySQL, you can write strings surrounded by either `'` or `"` characters (for example, `'hello'` or `"goodbye"`), and `mysql` lets you enter strings that span multiple lines. When you see a `'>` or `">` prompt, it means that you have entered a line containing a string that begins with a `'` or `"` quote character, but have not yet entered the matching quote that terminates the string. This often indicates that you have inadvertently left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
'>
```

If you enter this `SELECT` statement, then press **Enter** and wait for the result, nothing happens. Instead of wondering why this query takes so long, notice the clue provided by the `'>` prompt. It tells you that `mysql` expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `'Smith` is missing the second single quote mark.)

At this point, what do you do? The simplest thing is to cancel the command. However, you cannot just type `\c` in this case, because `mysql` interprets it as part of the string that it is collecting. Instead, enter the closing quote character (so `mysql` knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
'> \c
mysql>
```

The prompt changes back to `mysql>`, indicating that `mysql` is ready for a new command.

The `>` prompt is similar to the `'>` and `">` prompts, but indicates that you have begun but not completed a backtick-quoted identifier.

It is important to know what the `'>`, `">`, and `>` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type appear to be ignored by `mysql` — including a line containing `QUIT`. This can be quite confusing, especially if you do not know that you need to supply the terminating quote before you can cancel the current command.

3.3. Creating and Using a Database

Once you know how to enter commands, you are ready to access a database.

Suppose that you have several pets in your home (your menagerie) and you would like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to:

- Create a database
- Create a table
- Load data into the table
- Retrieve data from the table in various ways
- Use multiple tables

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be obtained from the MySQL Web site. It is available in both compressed `tar` file and Zip formats at <http://dev.mysql.com/doc/>.

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| mysql    |
| test     |
| tmp      |
+-----+
```

The `mysql` database describes user access privileges. The `test` database often is available as a workspace for users to try things out.

The list of databases displayed by the statement may be different on your machine; `SHOW DATABASES` does not show databases that you have no privileges for if you do not have the `SHOW DATABASES` privilege. See [Section 12.5.4.12, “SHOW DATABASES Syntax”](#).

If the `test` database exists, try to access it:

```
mysql> USE test
Database changed
```

Note that `USE`, like `QUIT`, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The `USE` statement is special in another way, too: it must be given on a single line.

You can use the `test` database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose that you want to call yours `menagerie`. The administrator needs to execute a command like this:

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

where `your_mysql_name` is the MySQL user name assigned to you and `your_client_host` is the host from which you connect to the server.

3.3.1. Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case sensitive (unlike SQL keywords), so you must always refer to your database as `menagerie`, not as `Menagerie`, `MENAGERIE`, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query. However, for a variety of reasons, our recommended best practice is always to use the same lettercase that was used when the database was created.)

Note

If you get an error such as `ERROR 1044 (42000): ACCESS DENIED FOR USER 'MONTY'@'LOCALHOST' TO DATABASE 'MENAGERIE'` when attempting to create a database, this means that your user account does not have the necessary privileges to do so. Discuss this with the administrator or see [Section 5.7, “The MySQL Access Privilege System”](#).

Creating a database does not select it for use; you must do that explicitly. To make `menagerie` the current database, use this command:

```
mysql> USE menagerie
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a `mysql` session. You can do this by issuing a `USE` statement as shown in the example. Alternatively, you can select the database on the command line when you invoke `mysql`. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie
Enter password: *****
```

Note that `menagerie` in the command just shown is **not** your password. If you want to supply your password on the command line after the `-p` option, you must do so with no intervening space (for example, as `-pmypassword`, not as `-p mypassword`). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

3.3.2. Creating a Table

Creating the database is the easy part, but at this point it's empty, as `SHOW TABLES` tells you:

```
mysql> SHOW TABLES;
```

```
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the `pet` table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it's not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it's better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you need to send out birthday greetings in the current week or month, for that computer-assisted personal touch.)
- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the `pet` table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
-> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the `name`, `owner`, and `species` columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be `20`. You can normally pick any length from `1` to `65535`, whatever seems most reasonable to you. If you make a poor choice and it turns out later that you need a longer field, MySQL provides an `ALTER TABLE` statement.

Several types of values can be chosen to represent sex in animal records, such as `'m'` and `'f'`, or perhaps `'male'` and `'female'`. It is simplest to use the single characters `'m'` and `'f'`.

The use of the `DATE` data type for the `birth` and `death` columns is a fairly obvious choice.

Once you have created a table, `SHOW TABLES` should produce some output:

```
mysql> SHOW TABLES;
+-----+
| Tables in menagerie |
+-----+
| pet                |
+-----+
```

To verify that your table was created the way you expected, use a `DESCRIBE` statement:

```
mysql> DESCRIBE pet;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | varchar(20) | YES  |     | NULL    |       |
| owner | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex   | char(1)    | YES  |     | NULL    |       |
| birth | date      | YES  |     | NULL    |       |
| death | date      | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

You can use `DESCRIBE` any time, for example, if you forget the names of the columns in your table or what types they have.

For more information about MySQL data types, see [Chapter 10, Data Types](#).

3.3.3. Loading Data into a Table

After creating your table, you need to populate it. The `LOAD DATA` and `INSERT` statements are useful for this.

Suppose that your pet records can be described as shown here. (Observe that MySQL expects dates in `'YYYY-MM-DD'` format; this may be different from what you are used to.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt` containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use `NULL` values. To represent these in your text file, use `\N` (backslash, capital-N). For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

```
Whistler      Gwen      bird      \N      1997-12-09      \N
```

To load the text file `pet.txt` into the `pet` table, use this command:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

Note that if you created the file on Windows with an editor that uses `\r\n` as a line terminator, you should use:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
-> LINES TERMINATED BY '\r\n';
```

(On an Apple machine running OS X, you would likely want to use `LINES TERMINATED BY '\r'`.)

You can specify the column value separator and end of line marker explicitly in the `LOAD DATA` statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file `pet.txt` properly.

If the statement fails, it is likely that your MySQL installation does not have local file capability enabled by default. See [Section 5.6.4, “Security Issues with `LOAD DATA LOCAL`”](#), for information on how to change this.

When you want to add new records one at a time, the `INSERT` statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the `CREATE TABLE` statement. Suppose that Diane gets a new hamster named “Puffball.” You could add a new record using an `INSERT` statement like this:

```
mysql> INSERT INTO pet
-> VALUES ('Puffball', 'Diane', 'hamster', 'f', '1999-03-30', NULL);
```

Note that string and date values are specified as quoted strings here. Also, with `INSERT`, you can insert `NULL` directly to represent a missing value. You do not use `\N` like you do with `LOAD DATA`.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several `INSERT` statements rather than a single `LOAD DATA` statement.

3.3.4. Retrieving Information from a Table

The **SELECT** statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

what_to_select indicates what you want to see. This can be a list of columns, or ***** to indicate “all columns.” *which_table* indicates the table from which you want to retrieve data. The **WHERE** clause is optional. If it is present, *conditions_to_satisfy* specifies one or more conditions that rows must satisfy to qualify for retrieval.

3.3.4.1. Selecting All Data

The simplest form of **SELECT** retrieves everything from a table:

```
mysql> SELECT * FROM pet;
```

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	NULL
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL
Fang	Benny	dog	m	1990-08-27	NULL
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	NULL
Whistler	Gwen	bird	NULL	1997-12-09	NULL
Slim	Benny	snake	m	1996-04-29	NULL
Puffball	Diane	hamster	f	1999-03-30	NULL

This form of **SELECT** is useful if you want to review your entire table, for example, after you've just loaded it with your initial dataset. For example, you may happen to think that the birth date for Bowser doesn't seem quite right. Consulting your original pedigree papers, you find that the correct birth year should be 1989, not 1979.

There are at least two ways to fix this:

- Edit the file `pet.txt` to correct the error, then empty the table and reload it using **DELETE** and **LOAD DATA**:

```
mysql> DELETE FROM pet;
mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
```

However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an **UPDATE** statement:

```
mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
```

The **UPDATE** changes only the record in question and does not require you to reload the table.

3.3.4.2. Selecting Particular Rows

As shown in the preceding section, it is easy to retrieve an entire table. Just omit the **WHERE** clause from the **SELECT** statement. But typically you don't want to see the entire table, particularly when it becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
```

name	owner	species	sex	birth	death
Bowser	Diane	dog	m	1989-08-31	1995-07-29

The output confirms that the year is correctly recorded as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as `'bowser'`, `'BOWSER'`, and so forth. The query result is the same.

You can specify conditions on any column, not just `name`. For example, if you want to know which animals were born during or after 1998, test the `birth` column:

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
```

name	owner	species	sex	birth	death
Chirpy	Gwen	bird	f	1998-09-11	NULL
Puffball	Diane	hamster	f	1999-03-30	NULL

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
```

name	owner	species	sex	birth	death
Buffy	Harold	dog	f	1989-05-13	NULL

The preceding query uses the `AND` logical operator. There is also an `OR` operator:

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
```

name	owner	species	sex	birth	death
Chirpy	Gwen	bird	f	1998-09-11	NULL
Whistler	Gwen	bird	NULL	1997-12-09	NULL
Slim	Benny	snake	m	1996-04-29	NULL

`AND` and `OR` may be intermixed, although `AND` has higher precedence than `OR`. If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
-> OR (species = 'dog' AND sex = 'f');
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

3.3.4.3. Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas. For example, if you want to know when your animals were born, select the `name` and `birth` columns:

```
mysql> SELECT name, birth FROM pet;
```

name	birth
Fluffy	1993-02-04
Claws	1994-03-17
Buffy	1989-05-13
Fang	1990-08-27
Bowser	1989-08-31
Chirpy	1998-09-11
Whistler	1997-12-09
Slim	1996-04-29
Puffball	1999-03-30

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
```

owner
Harold
Gwen
Harold
Benny

```

| Diane |
| Gwen |
| Gwen |
| Benny |
| Diane |
+-----+

```

Notice that the query simply retrieves the `owner` column from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```

mysql> SELECT DISTINCT owner FROM pet;
+-----+
| owner |
+-----+
| Benny |
| Diane |
| Gwen |
| Harold |
+-----+

```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```

mysql> SELECT name, species, birth FROM pet
-> WHERE species = 'dog' OR species = 'cat';
+-----+-----+-----+
| name | species | birth |
+-----+-----+-----+
| Fluffy | cat | 1993-02-04 |
| Claws | cat | 1994-03-17 |
| Buffy | dog | 1989-05-13 |
| Fang | dog | 1990-08-27 |
| Bowser | dog | 1989-08-31 |
+-----+-----+-----+

```

3.3.4.4. Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. It's often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```

mysql> SELECT name, birth FROM pet ORDER BY birth;
+-----+-----+
| name | birth |
+-----+-----+
| Buffy | 1989-05-13 |
| Bowser | 1989-08-31 |
| Fang | 1990-08-27 |
| Fluffy | 1993-02-04 |
| Claws | 1994-03-17 |
| Slim | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy | 1998-09-11 |
| Puffball | 1999-03-30 |
+-----+-----+

```

On character type columns, sorting — like all other comparison operations — is normally performed in a case-insensitive fashion. This means that the order is undefined for columns that are identical except for their case. You can force a case-sensitive sort for a column by using `BINARY` like so: `ORDER BY BINARY col_name`.

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `DESC` keyword to the name of the column you are sorting by:

```

mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+-----+-----+
| name | birth |
+-----+-----+
| Puffball | 1999-03-30 |
| Chirpy | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim | 1996-04-29 |
| Claws | 1994-03-17 |
| Fluffy | 1993-02-04 |
| Fang | 1990-08-27 |
| Bowser | 1989-08-31 |
| Buffy | 1989-05-13 |
+-----+-----+

```

You can sort on multiple columns, and you can sort different columns in different directions. For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet
-> ORDER BY species, birth DESC;
```

name	species	birth
Chirpy	bird	1998-09-11
Whistler	bird	1997-12-09
Claws	cat	1994-03-17
Fluffy	cat	1993-02-04
Fang	dog	1990-08-27
Bowser	dog	1989-08-31
Buffy	dog	1989-05-13
Puffball	hamster	1999-03-30
Slim	snake	1996-04-29

Note that the **DESC** keyword applies only to the column name immediately preceding it (**birth**); it does not affect the **species** column sort order.

3.3.4.5. Date Calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, compute the difference in the year part of the current date and the birth date, then subtract one if the current date occurs earlier in the calendar year than the birth date. The following query shows, for each pet, the birth date, the current date, and the age in years.

```
mysql> SELECT name, birth, CURDATE(),
-> (YEAR(CURDATE())-YEAR(birth))
-> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
-> AS age
-> FROM pet;
```

name	birth	CURDATE()	age
Fluffy	1993-02-04	2003-08-19	10
Claws	1994-03-17	2003-08-19	9
Buffy	1989-05-13	2003-08-19	14
Fang	1990-08-27	2003-08-19	12
Bowser	1989-08-31	2003-08-19	13
Chirpy	1998-09-11	2003-08-19	4
Whistler	1997-12-09	2003-08-19	5
Slim	1996-04-29	2003-08-19	7
Puffball	1999-03-30	2003-08-19	4

Here, **YEAR()** pulls out the year part of a date and **RIGHT()** pulls off the rightmost five characters that represent the **MM-DD** (calendar year) part of the date. The part of the expression that compares the **MM-DD** values evaluates to 1 or 0, which adjusts the year difference down a year if **CURDATE()** occurs earlier in the year than **birth**. The full expression is somewhat ungainly, so an *alias* (**age**) is used to make the output column label more meaningful.

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an **ORDER BY name** clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
-> (YEAR(CURDATE())-YEAR(birth))
-> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
-> AS age
-> FROM pet ORDER BY name;
```

name	birth	CURDATE()	age
Bowser	1989-08-31	2003-08-19	13
Buffy	1989-05-13	2003-08-19	14
Chirpy	1998-09-11	2003-08-19	4
Claws	1994-03-17	2003-08-19	9
Fang	1990-08-27	2003-08-19	12
Fluffy	1993-02-04	2003-08-19	10
Puffball	1999-03-30	2003-08-19	4
Slim	1996-04-29	2003-08-19	7
Whistler	1997-12-09	2003-08-19	5

To sort the output by `age` rather than `name`, just use a different `ORDER BY` clause:

```
mysql> SELECT name, birth, CURDATE(),
-> (YEAR(CURDATE())-YEAR(birth))
-> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
-> AS age
-> FROM pet ORDER BY age;
```

name	birth	CURDATE()	age
Chirpy	1998-09-11	2003-08-19	4
Puffball	1999-03-30	2003-08-19	4
Whistler	1997-12-09	2003-08-19	5
Slim	1996-04-29	2003-08-19	7
Claws	1994-03-17	2003-08-19	9
Fluffy	1993-02-04	2003-08-19	10
Fang	1990-08-27	2003-08-19	12
Bowser	1989-08-31	2003-08-19	13
Buffy	1989-05-13	2003-08-19	14

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the `death` value is `NULL`. Then, for those with non-`NULL` values, compute the difference between the `death` and `birth` values:

```
mysql> SELECT name, birth, death,
-> (YEAR(death)-YEAR(birth)) - (RIGHT(death,5)<RIGHT(birth,5))
-> AS age
-> FROM pet WHERE death IS NOT NULL ORDER BY age;
```

name	birth	death	age
Bowser	1989-08-31	1995-07-29	5

The query uses `death IS NOT NULL` rather than `death <> NULL` because `NULL` is a special value that cannot be compared using the usual comparison operators. This is discussed later. See [Section 3.3.4.6, “Working with NULL Values”](#).

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the `birth` column. MySQL provides several functions for extracting parts of dates, such as `YEAR()`, `MONTH()`, and `DAYOFMONTH()`. `MONTH()` is the appropriate function here. To see how it works, run a simple query that displays the value of both `birth` and `MONTH(birth)`:

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
```

name	birth	MONTH(birth)
Fluffy	1993-02-04	2
Claws	1994-03-17	3
Buffy	1989-05-13	5
Fang	1990-08-27	8
Bowser	1989-08-31	8
Chirpy	1998-09-11	9
Whistler	1997-12-09	12
Slim	1996-04-29	4
Puffball	1999-03-30	3

Finding animals with birthdays in the upcoming month is also simple. Suppose that the current month is April. Then the month value is `4` and you can look for animals born in May (month `5`) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```

name	birth
Buffy	1989-05-13

There is a small complication if the current month is December. You cannot merely add one to the month number (`12`) and look for animals born in month `13`, because there is no such month. Instead, you look for animals born in January (month `1`).

You can write the query so that it works no matter what the current month is, so that you do not have to use the number for a particular month. `DATE_ADD()` allows you to add a time interval to a given date. If you add a month to the value of `CURDATE()`, then extract the month part with `MONTH()`, the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
```

```
-> WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

A different way to accomplish the same task is to add **1** to get the next month after the current one after using the modulo function (**MOD**) to wrap the month value to **0** if it is currently **12**:

```
mysql> SELECT name, birth FROM pet
-> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

Note that **MONTH** returns a number between **1** and **12**. And **MOD(something, 12)** returns a number between **0** and **11**. So the addition has to be after the **MOD()**, otherwise we would go from November (**11**) to January (**1**).

3.3.4.6. Working with NULL Values

The **NULL** value can be surprising until you get used to it. Conceptually, **NULL** means “a missing unknown value” and it is treated somewhat differently from other values. To test for **NULL**, you cannot use the arithmetic comparison operators such as **=**, **<**, or **<>**. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+-----+-----+-----+-----+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+-----+-----+-----+-----+
| NULL | NULL | NULL | NULL |
+-----+-----+-----+-----+
```

Clearly you get no meaningful results from these comparisons. Use the **IS NULL** and **IS NOT NULL** operators instead:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----+-----+
| 1 IS NULL | 1 IS NOT NULL |
+-----+-----+
| 0 | 1 |
+-----+-----+
```

Note that in MySQL, **0** or **NULL** means false and anything else means true. The default truth value from a boolean operation is **1**.

This special treatment of **NULL** is why, in the previous section, it was necessary to determine which animals are no longer alive using **death IS NOT NULL** instead of **death <> NULL**.

Two **NULL** values are regarded as equal in a **GROUP BY**.

When doing an **ORDER BY**, **NULL** values are presented first if you do **ORDER BY ... ASC** and last if you do **ORDER BY ... DESC**.

A common error when working with **NULL** is to assume that it is not possible to insert a zero or an empty string into a column defined as **NOT NULL**, but this is not the case. These are in fact values, whereas **NULL** means “not having a value.” You can test this easily enough by using **IS [NOT] NULL** as shown:

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----+-----+-----+-----+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----+-----+-----+-----+
| 0 | 1 | 0 | 1 |
+-----+-----+-----+-----+
```

Thus it is entirely possible to insert a zero or empty string into a **NOT NULL** column, as these are in fact **NOT NULL**. See [Section B.1.5.3, “Problems with NULL Values”](#).

3.3.4.7. Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as **vi**, **grep**, and **sed**.

SQL pattern matching allows you to use **'_'** to match any single character and **'%'** to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. Note that you do not use **=** or **<>** when you use SQL patterns; use the **LIKE** or **NOT LIKE** comparison operators instead.

To find names beginning with **'b'**:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
```

name	owner	species	sex	birth	death
Buffy	Harold	dog	f	1989-05-13	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29

To find names ending with 'fy':

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
```

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

To find names containing a 'w':

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29
Whistler	Gwen	bird	NULL	1997-12-09	NULL

To find names containing exactly five characters, use five instances of the '_' pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

The other type of pattern matching provided by MySQL uses extended regular expressions. When you test for a match for this type of pattern, use the **REGEXP** and **NOT REGEXP** operators (or **RLIKE** and **NOT RLIKE**, which are synonyms).

Some characteristics of extended regular expressions are:

- '.' matches any single character.
- A character class '[...]' matches any character within the brackets. For example, '[abc]' matches 'a', 'b', or 'c'. To name a range of characters, use a dash. '[a-z]' matches any letter, whereas '[0-9]' matches any digit.
- '*' matches zero or more instances of the thing preceding it. For example, 'x*' matches any number of 'x' characters, '[0-9]*' matches any number of digits, and '.*' matches any number of anything.
- A **REGEXP** pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a **LIKE** pattern match, which succeeds only if the pattern matches the entire value.)
- To anchor a pattern so that it must match the beginning or end of the value being tested, use '^' at the beginning or '\$' at the end of the pattern.

To demonstrate how extended regular expressions work, the **LIKE** queries shown previously are rewritten here to use **REGEXP**.

To find names beginning with 'b', use '^' to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
```

name	owner	species	sex	birth	death
Buffy	Harold	dog	f	1989-05-13	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29

If you really want to force a [REGEXP](#) comparison to be case sensitive, use the [BINARY](#) keyword to make one of the strings a binary string. This query matches only lowercase ‘b’ at the beginning of a name:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

To find names ending with ‘fy’, use ‘\$’ to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
```

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

To find names containing a ‘w’, use this query:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29
Whistler	Gwen	bird	NULL	1997-12-09	NULL

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value like it would be if you used an SQL pattern.

To find names containing exactly five characters, use ‘^’ and ‘\$’ to match the beginning and end of the name, and five instances of ‘.’ in between:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.....$';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

You could also write the previous query using the `{n}` (“repeat-*n*-times”) operator:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.{5}$';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

[Section 11.4.2, “Regular Expressions”](#), provides more information about the syntax for regular expressions.

3.3.4.8. Counting Rows

Databases are often used to answer the question, “How often does a certain type of data occur in a table?” For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

Counting the total number of animals you have is the same question as “How many rows are in the `pet` table?” because there is one record per pet. `COUNT(*)` counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
```

COUNT(*)
9

Earlier, you retrieved the names of the people who owned pets. You can use `COUNT()` if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
```

owner	COUNT(*)
Benny	2
Diane	2
Gwen	3
Harold	2

Note the use of **GROUP BY** to group all records for each **owner**. Without it, all you get is an error message:

```
mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT(),...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

COUNT() and **GROUP BY** are useful for characterizing your data in various ways. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
```

species	COUNT(*)
bird	2
cat	2
dog	3
hamster	1
snake	1

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
```

sex	COUNT(*)
NULL	1
f	4
m	4

(In this output, **NULL** indicates that the sex is unknown.)

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
```

species	sex	COUNT(*)
bird	NULL	1
bird	f	1
cat	f	1
cat	m	1
dog	f	1
dog	m	2
hamster	f	1
snake	m	1

You need not retrieve an entire table when you use **COUNT()**. For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
-> WHERE species = 'dog' OR species = 'cat'
-> GROUP BY species, sex;
```

species	sex	COUNT(*)
cat	f	1
cat	m	1
dog	f	1
dog	m	2

Or, if you wanted the number of animals per sex only for animals whose sex is known:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
-> WHERE sex IS NOT NULL
-> GROUP BY species, sex;
```

species	sex	COUNT(*)
bird	f	1
cat	f	1
cat	m	1
dog	f	1
dog	m	2
hamster	f	1
snake	m	1

3.3.4.9. Using More Than one Table

The `pet` table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs:

- To contain the pet name so that you know which animal each event pertains to.
- A date so that you know when the event occurred.
- A field to describe the event.
- An event type field, if you want to be able to categorize events.

Given these considerations, the `CREATE TABLE` statement for the `event` table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
-> type VARCHAR(15), remark VARCHAR(255));
```

As with the `pet` table, it's easiest to load the initial records by creating a tab-delimited text file containing the information:

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Based on what you have learned from the queries that you have run on the `pet` table, you should be able to perform retrievals on the records in the `event` table; the principles are the same. But when is the `event` table by itself insufficient to answer questions you might ask?

Suppose that you want to find out the ages at which each pet had its litters. We saw earlier how to calculate ages from two dates. The litter date of the mother is in the `event` table, but to calculate her age on that date you need her birth date, which is stored in the `pet` table. This means the query requires both tables:

```
mysql> SELECT pet.name,
-> (YEAR(date)-YEAR(birth)) - (RIGHT(date,5)<RIGHT(birth,5)) AS age,
-> remark
-> FROM pet INNER JOIN event
-> ON pet.name = event.name
-> WHERE event.type = 'litter';
```

name	age	remark
Fluffy	2	4 kittens, 3 female, 1 male
Buffy	4	5 puppies, 2 female, 3 male
Buffy	5	3 puppies, 3 female

There are several things to note about this query:

- The **FROM** clause joins two tables because the query needs to pull information from both of them.
- When combining (joining) information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy because they both have a **name** column. The query uses an **ON** clause to match up records in the two tables based on the **name** values.

The query uses an **INNER JOIN** to combine the tables. An **INNER JOIN** allows for rows from either table to appear in the result if and only if both tables meet the conditions specified in the **ON** clause. In this example, the **ON** clause specifies that the **name** column in the **pet** table must match the **name** column in the **event** table. If a name appears in one table but not the other, the row will not appear in the result because the condition in the **ON** clause fails.

- Because the **name** column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the **pet** table with itself to produce candidate pairs of males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
-> FROM pet AS p1 INNER JOIN pet AS p2
-> ON p1.species = p2.species AND p1.sex = 'f' AND p2.sex = 'm';
```

name	sex	name	sex	species
Fluffy	f	Claws	m	cat
Buffy	f	Fang	m	dog
Buffy	f	Bowser	m	dog

In this query, we specify aliases for the table name to refer to the columns and keep straight which instance of the table each column reference is associated with.

3.4. Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

You have previously seen **SHOW DATABASES**, which lists the databases managed by the server. To find out which database is currently selected, use the **DATABASE()** function:

```
mysql> SELECT DATABASE();
```

DATABASE()
menagerie

If you have not yet selected any database, the result is **NULL**.

To find out what tables the default database contains (for example, when you are not sure about the name of a table), use this command:

```
mysql> SHOW TABLES;
```

Tables_in_menagerie

```
+-----+
| event |
| pet   |
+-----+
```

The name of the column in the output produced by this statement is always `Tables_in_db_name`, where `db_name` is the name of the database. See [Section 12.5.4.29, “SHOW TABLES Syntax”](#), for more information.

If you want to find out about the structure of a table, the `DESCRIBE` command is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | varchar(20) | YES  |     | NULL    |       |
| owner | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex   | char(1)    | YES  |     | NULL    |       |
| birth | date       | YES  |     | NULL    |       |
| death | date       | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

`Field` indicates the column name, `Type` is the data type for the column, `NULL` indicates whether the column can contain `NULL` values, `Key` indicates whether the column is indexed, and `Default` specifies the column's default value. `Extra` displays special information about columns; for example, if a column was created with the `AUTO_INCREMENT` option, this is shown here.

`DESC` is a short form of `DESCRIBE`. See [Section 12.3.1, “DESCRIBE Syntax”](#), for more information.

You can obtain the `CREATE TABLE` statement necessary to create an existing table using the `SHOW CREATE TABLE` statement. See [Section 12.5.4.9, “SHOW CREATE TABLE Syntax”](#).

If you have indexes on a table, `SHOW INDEX FROM tbl_name` produces information about them. See [Section 12.5.4.18, “SHOW INDEX Syntax”](#), for more about this statement.

3.5. Using `mysql` in Batch Mode

In the previous sections, you used `mysql` interactively to enter queries and view the results. You can also run `mysql` in batch mode. To do this, put the commands you want to run in a file, then tell `mysql` to read its input from the file:

```
shell> mysql < batch-file
```

If you are running `mysql` under Windows and have some special characters in the file that cause problems, you can do this:

```
C:\> mysql -e "source batch-file"
```

If you need to specify connection parameters on the command line, the command might look like this:

```
shell> mysql -h host -u user -p < batch-file
Enter password: *****
```

When you use `mysql` this way, you are creating a script file, then executing the script.

If you want the script to continue even if some of the statements in it produce errors, you should use the `--force` command-line option.

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script allows you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line commands or multiple-statement sequences of commands. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell `mysql` to execute it again.

- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

```
shell> mysql < batch-file | more
```

- You can catch the output in a file for further processing:

```
shell> mysql < batch-file > mysql.out
```

- You can distribute your script to other people so that they can also run the commands.
- Some situations do not allow for interactive use, for example, when you run a query from a [cron](#) job. In this case, you must use batch mode.

The default output format is different (more concise) when you run `mysql` in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when `mysql` is run interactively:

```
+-----+
| species |
+-----+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+-----+
```

In batch mode, the output looks like this instead:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the commands that are executed, use `mysql -vvv`.

You can also use scripts from the `mysql` prompt by using the `source` command or `\.` command:

```
mysql> source filename;
mysql> \. filename
```

See [Section 7.8.4, “Executing SQL Statements from a Text File”](#), for more information.

3.6. Examples of Common Queries

Here are examples of how to solve some common problems with MySQL.

Some of the examples use the table `shop` to hold the price of each article (item number) for certain traders (dealers). Supposing that each trader has a single fixed price per article, then `(article, dealer)` is a primary key for the records.

Start the command-line tool `mysql` and select a database:

```
shell> mysql your-database-name
```

(In most MySQL installations, you can use the database named `test`).

You can create and populate the example table with these statements:

```
CREATE TABLE shop (
  article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
  dealer CHAR(20) DEFAULT '' NOT NULL,
  price DOUBLE(16,2) DEFAULT '0.00' NOT NULL,
  PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
```

```
(1, 'A', 3.45), (1, 'B', 3.99), (2, 'A', 10.99), (3, 'B', 1.45),
(3, 'C', 1.69), (3, 'D', 1.25), (4, 'D', 19.95);
```

After issuing the statements, the table should have the following contents:

```
SELECT * FROM shop;
```

article	dealer	price
0001	A	3.45
0001	B	3.99
0002	A	10.99
0003	B	1.45
0003	C	1.69
0003	D	1.25
0004	D	19.95

3.6.1. The Maximum Value for a Column

“What’s the highest item number?”

```
SELECT MAX(article) AS article FROM shop;
```

article
4

3.6.2. The Row Holding the Maximum of a Certain Column

Task: Find the number, dealer, and price of the most expensive article.

This is easily done with a subquery:

```
SELECT article, dealer, price
FROM shop
WHERE price=(SELECT MAX(price) FROM shop);
```

Another solution is to sort all rows descending by price and get only the first row using the MySQL-specific [LIMIT](#) clause:

```
SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

Note

If there were several most expensive articles, each with a price of 19.95, the [LIMIT](#) solution would show only one of them.

3.6.3. Maximum of Column per Group

Task: Find the highest price per article.

```
SELECT article, MAX(price) AS price
FROM shop
GROUP BY article
```

article	price
0001	3.99
0002	10.99
0003	1.69
0004	19.95

3.6.4. The Rows Holding the Group-wise Maximum of a Certain Field

Task: For each article, find the dealer or dealers with the most expensive price.

This problem can be solved with a subquery like this one:

```
SELECT article, dealer, price
FROM shop s1
WHERE price=(SELECT MAX(s2.price)
             FROM shop s2
             WHERE s1.article = s2.article);
```

article	dealer	price
0001	B	3.99
0002	A	10.99
0003	C	1.69
0004	D	19.95

The preceding example uses a correlated subquery, which can be inefficient (see [Section 12.2.8.7, “Correlated Subqueries”](#)). Other possibilities for solving the problem are to use an uncorrelated subquery in the `FROM` clause or a `LEFT JOIN`:

```
SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
ON s1.article = s2.article AND s1.price = s2.price;

SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
WHERE s2.article IS NULL;
```

The `LEFT JOIN` works on the basis that when `s1.price` is at its maximum value, there is no `s2.price` with a greater value and the `s2` rows values will be `NULL`. See [Section 12.2.7.1, “JOIN Syntax”](#).

3.6.5. Using User-Defined Variables

You can employ MySQL user variables to remember results without having to store them in temporary variables in the client. (See [Section 8.4, “User-Defined Variables”](#).)

For example, to find the articles with the highest and lowest price you can do this:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
```

article	dealer	price
0003	D	1.25
0004	D	19.95

3.6.6. Using Foreign Keys

In MySQL, `InnoDB` tables support checking of foreign key constraints. See [Section 13.5, “The InnoDB Storage Engine”](#), and [Section 1.9.5.4, “Foreign Keys”](#).

A foreign key constraint is not required merely to join two tables. For storage engines other than `InnoDB`, it is possible when defining a column to use a `REFERENCES tbl_name(col_name)` clause, which has no actual effect, and serves only as a memo or comment to you that the column which you are currently defining is intended to refer to a column in another table. It is extremely important to realize when using this syntax that:

- MySQL does not perform any sort of `CHECK` to make sure that `col_name` actually exists in `tbl_name` (or even that `tbl_name` itself exists).
- MySQL does not perform any sort of action on `tbl_name` such as deleting rows in response to actions taken on rows in the table which you are defining; in other words, this syntax induces no `ON DELETE` or `ON UPDATE` behavior whatsoever. (Although you

can write an **ON DELETE** or **ON UPDATE** clause as part of the **REFERENCES** clause, it is also ignored.)

- This syntax creates a *column*; it does **not** create any sort of index or key.
- This syntax will cause an error if used in trying to define an **InnoDB** table.

You can use a column so created as a join column, as shown here:

```
CREATE TABLE person (
  id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  name CHAR(60) NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE shirt (
  id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
  color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
  owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
  PRIMARY KEY (id)
);

INSERT INTO person VALUES (NULL, 'Antonio Paz');
SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');
SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+-----+
| id | name          |
+-----+
| 1  | Antonio Paz   |
| 2  | Lilliana Angelovska |
+-----+

SELECT * FROM shirt;
+-----+-----+-----+-----+
| id | style | color | owner |
+-----+-----+-----+-----+
| 1  | polo  | blue  | 1      |
| 2  | dress | white | 1      |
| 3  | t-shirt | blue  | 1      |
| 4  | dress | orange | 2      |
| 5  | polo  | red    | 2      |
| 6  | dress | blue   | 2      |
| 7  | t-shirt | white  | 2      |
+-----+-----+-----+-----+

SELECT s.* FROM person p INNER JOIN shirt s
  ON s.owner = p.id
 WHERE p.name LIKE 'Lilliana%'
    AND s.color <> 'white';

+-----+-----+-----+-----+
| id | style | color | owner |
+-----+-----+-----+-----+
| 4  | dress | orange | 2      |
| 5  | polo  | red    | 2      |
| 6  | dress | blue   | 2      |
+-----+-----+-----+-----+
```

When used in this fashion, the **REFERENCES** clause is not displayed in the output of **SHOW CREATE TABLE** or **DESCRIBE**:

```
SHOW CREATE TABLE shirt\G
***** 1. row *****
Table: shirt
Create Table: CREATE TABLE `shirt` (
  `id` smallint(5) unsigned NOT NULL auto_increment,
  `style` enum('t-shirt','polo','dress') NOT NULL,
```

```
`color` enum('red','blue','orange','white','black') NOT NULL,
`owner` smallint(5) unsigned NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

The use of [REFERENCES](#) in this way as a comment or “reminder” in a column definition works with [MyISAM](#) tables.

3.6.7. Searching on Two Keys

An [OR](#) using a single key is well optimized, as is the handling of [AND](#).

The one tricky case is that of searching on two different keys combined with [OR](#):

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR field2_index = '1'
```

This case is optimized. See [Section 6.2.6, “Index Merge Optimization”](#).

You can also solve the problem efficiently by using a [UNION](#) that combines the output of two separate [SELECT](#) statements. See [Section 12.2.7.3, “UNION Syntax”](#).

Each [SELECT](#) searches only one key and can be optimized:

```
SELECT field1_index, field2_index
FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
FROM test_table WHERE field2_index = '1';
```

3.6.8. Calculating Visits Per Day

The following example shows how you can use the bit group functions to calculate the number of days per month a user has visited a Web page.

```
CREATE TABLE t1 (year YEAR(4), month INT(2) UNSIGNED ZEROFILL,
day INT(2) UNSIGNED ZEROFILL);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
(2000,2,23),(2000,2,23);
```

The example table contains year-month-day values representing visits by users to the page. To determine how many different days in each month these visits occur, use this query:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
GROUP BY year,month;
```

Which returns:

year	month	days
2000	01	3
2000	02	2

The query calculates how many different days appear in the table for each year/month combination, with automatic removal of duplicate entries.

3.6.9. Using [AUTO_INCREMENT](#)

The [AUTO_INCREMENT](#) attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (
id MEDIUMINT NOT NULL AUTO_INCREMENT,
name CHAR(30) NOT NULL,
PRIMARY KEY (id)
);
INSERT INTO animals (name) VALUES
```

```

('dog'),('cat'),('penguin'),
('lax'),('whale'),('ostrich');
SELECT * FROM animals;

```

Which returns:

id	name
1	dog
2	cat
3	penguin
4	lax
5	whale
6	ostrich

You can retrieve the most recent [AUTO_INCREMENT](#) value with the [LAST_INSERT_ID\(\)](#) SQL function or the [mysql_insert_id\(\)](#) C API function. These functions are connection-specific, so their return values are not affected by another connection which is also performing inserts.

Note

For a multiple-row insert, [LAST_INSERT_ID\(\)](#) and [mysql_insert_id\(\)](#) actually return the [AUTO_INCREMENT](#) key from the *first* of the inserted rows. This allows multiple-row inserts to be reproduced correctly on other servers in a replication setup.

For [MyISAM](#) tables you can specify [AUTO_INCREMENT](#) on a secondary column in a multiple-column index. In this case, the generated value for the [AUTO_INCREMENT](#) column is calculated as [MAX\(auto_increment_column\) + 1 WHERE prefix=given-prefix](#). This is useful when you want to put data into ordered groups.

```

CREATE TABLE animals (
  grp ENUM('fish','mammal','bird') NOT NULL,
  id MEDIUMINT NOT NULL AUTO_INCREMENT,
  name CHAR(30) NOT NULL,
  PRIMARY KEY (grp,id)
);

INSERT INTO animals (grp,name) VALUES
('mammal','dog'),('mammal','cat'),
('bird','penguin'),('fish','lax'),('mammal','whale'),
('bird','ostrich');

SELECT * FROM animals ORDER BY grp,id;

```

Which returns:

grp	id	name
fish	1	lax
mammal	1	dog
mammal	2	cat
mammal	3	whale
bird	1	penguin
bird	2	ostrich

Note that in this case (when the [AUTO_INCREMENT](#) column is part of a multiple-column index), [AUTO_INCREMENT](#) values are re-used if you delete the row with the biggest [AUTO_INCREMENT](#) value in any group. This happens even for [MyISAM](#) tables, for which [AUTO_INCREMENT](#) values normally are not reused.

If the [AUTO_INCREMENT](#) column is part of multiple indexes, MySQL will generate sequence values using the index that begins with the [AUTO_INCREMENT](#) column, if there is one. For example, if the [animals](#) table contained indexes [PRIMARY KEY \(grp, id\)](#) and [INDEX \(id\)](#), MySQL would ignore the [PRIMARY KEY](#) for generating sequence values. As a result, the table would contain a single sequence, not a sequence per [grp](#) value.

To start with an [AUTO_INCREMENT](#) value other than 1, you can set that value with [CREATE TABLE](#) or [ALTER TABLE](#), like this:

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

More information about [AUTO_INCREMENT](#) is available here:

- How to assign the [AUTO_INCREMENT](#) attribute to a column: [Section 12.1.10, “CREATE TABLE Syntax”](#), and [Section 12.1.4, “ALTER TABLE Syntax”](#).
- How [AUTO_INCREMENT](#) behaves depending on the SQL mode: [Section 5.2.6, “SQL Modes”](#).
- Find the row that contains the most recent [AUTO_INCREMENT](#) value: [Section 11.2.3, “Comparison Functions and Operators”](#).
- Set the [AUTO_INCREMENT](#) value to be used: [Section 12.5.3, “SET Syntax”](#).
- [AUTO_INCREMENT](#) and replication: [Section 15.3.1, “Replication Features and Issues”](#).
- Server-system variables related to [AUTO_INCREMENT](#) ([auto_increment_increment](#) and [auto_increment_offset](#)) that can be used for replication: [Section 5.2.3, “System Variables”](#).

3.7. Queries from the Twin Project

At the places the early MySQL was developed (Analytikerna and Lentus), the founders did systems and field work for a big research project. This project was a collaboration between the Institute of Environmental Medicine at Karolinska Institutet Stockholm and the Section on Clinical Research in Aging and Psychology at the University of Southern California.

The project involved lots of data collection from all twins in Sweden older than 65 Years (see http://www.mep.ki.se/twinreg/index_en.html).

Large parts of the project were administered with a Web interface written using Perl and MySQL.

3.7.1. Find All Non-distributed Twins

The following query was used to determine what twins should be studied further after a initial screening. The time for this was around MySQL 3.19 in 1997.

```
SELECT
  CONCAT(p1.id, p1.tvab) + 0 AS tvid,
  CONCAT(p1.christian_name, ' ', p1.surname) AS Name,
  p1.postal_code AS Code,
  p1.city AS City,
  pg.abrev AS Area,
  IF(td.participation = 'Aborted', 'A', ' ') AS A,
  p1.dead AS dead1,
  l.event AS event1,
  td.suspect AS tsuspect1,
  id.suspect AS isuspect1,
  td.severe AS tsevere1,
  id.severe AS isevere1,
  p2.dead AS dead2,
  l2.event AS event2,
  h2.nurse AS nurse2,
  h2.doctor AS doctor2,
  td2.suspect AS tsuspect2,
  id2.suspect AS isuspect2,
  td2.severe AS tsevere2,
  id2.severe AS isevere2,
  l.finish_date
FROM
  twin_project AS tp
  /* For Twin 1 */
  LEFT JOIN twin_data AS td ON tp.id = td.id
    AND tp.tvab = td.tvab
  LEFT JOIN informant_data AS id ON tp.id = id.id
    AND tp.tvab = id.tvab
  LEFT JOIN harmony AS h ON tp.id = h.id
    AND tp.tvab = h.tvab
  LEFT JOIN lentus AS l ON tp.id = l.id
    AND tp.tvab = l.tvab
  /* For Twin 2 */
  LEFT JOIN twin_data AS td2 ON p2.id = td2.id
    AND p2.tvab = td2.tvab
  LEFT JOIN informant_data AS id2 ON p2.id = id2.id
    AND p2.tvab = id2.tvab
  LEFT JOIN harmony AS h2 ON p2.id = h2.id
    AND p2.tvab = h2.tvab
  LEFT JOIN lentus AS l2 ON p2.id = l2.id
    AND p2.tvab = l2.tvab,
```

```

person_data AS p1,
person_data AS p2,
postal_groups AS pg
WHERE
/* p1 gets main twin and p2 gets his/her twin. */
/* ptvab is a field inverted from tvab */
p1.id = tp.id AND p1.tvab = tp.tvab AND
p2.id = p1.id AND p2.ptvab = p1.tvab AND
/* Just the screening survey */
tp.survey_no = 5 AND
/* Skip if partner died before 65 but allow emigration (dead=9) */
(p2.dead = 0 OR p2.dead = 9 OR
 (p2.dead = 1 AND
  (p2.death_date = 0 OR
   (((TO_DAYS(p2.death_date) - TO_DAYS(p2.birthday)) / 365)
    >= 65))))
AND
(
/* Twin is suspect */
(td.future_contact = 'Yes' AND td.suspect = 2) OR
/* Twin is suspect - Informant is Blessed */
(td.future_contact = 'Yes' AND td.suspect = 1
 AND id.suspect = 1) OR
/* No twin - Informant is Blessed */
(ISNULL(td.suspect) AND id.suspect = 1
 AND id.future_contact = 'Yes') OR
/* Twin broken off - Informant is Blessed */
(td.participation = 'Aborted'
 AND id.suspect = 1 AND id.future_contact = 'Yes') OR
/* Twin broken off - No inform - Have partner */
(td.participation = 'Aborted' AND ISNULL(id.suspect)
 AND p2.dead = 0))
AND
l.event = 'Finished'
/* Get at area code */
AND SUBSTRING(p1.postal_code, 1, 2) = pg.code
/* Not already distributed */
AND (h.nurse IS NULL OR h.nurse=00 OR h.doctor=00)
/* Has not refused or been aborted */
AND NOT (h.status = 'Refused' OR h.status = 'Aborted'
 OR h.status = 'Died' OR h.status = 'Other')
ORDER BY
tvid;

```

Some explanations:

- `CONCAT(p1.id, p1.tvab) + 0 AS tvid`

We want to sort on the concatenated `id` and `tvab` in numerical order. Adding `0` to the result causes MySQL to treat the result as a number.

- column `id`

This identifies a pair of twins. It is an index in all tables.

- column `tvab`

This identifies a twin in a pair. It has a value of `1` or `2`.

- column `ptvab`

This is an inverse of `tvab`. When `tvab` is `1` this is `2`, and vice versa. It exists to save typing and to make it easier for MySQL to optimize the query.

This query demonstrates, among other things, how to do lookups on a table from the same table with a join (`p1` and `p2`). In the example, this is used to check whether a twin's partner died before the age of 65. If so, the row is not returned.

All of the above exist in all tables with twin-related information. We have an index on both `id`, `tvab` (all tables), and `id`, `ptvab` (`person_data`) to make queries faster.

When we did this work, our production machine was a 200MHz UltraSPARC, and on that old hardware this query returned about 150-200 rows in less than one second. The main table had 70k Rows.

3.7.2. Show a Table of Twin Pair Status

Each twin has a status code called [event](#). The query shown here is used to select all twin pairs combined by event. This indicates in how many pairs both twins are finished, in how many pairs one twin is finished and the other refused, and so on.

```
SELECT      t1.event,
            t2.event,
            COUNT(*)
FROM        lentus AS t1,
            lentus AS t2,
            twin_project AS tp
WHERE       /* We are looking at one pair at a time */
            t1.id = tp.id
            AND t1.tvab=tp.tvab
            AND t1.id = t2.id
            /* Just the screening survey */
            AND tp.survey_no = 5
            /* This makes each pair only appear once */
            AND t1.tvab='1' AND t2.tvab='2'
GROUP BY    t1.event, t2.event;
```

3.8. Using MySQL with Apache

There are programs that let you authenticate your users from a MySQL database and also let you write your log files into a MySQL table.

You can change the Apache logging format to be easily readable by MySQL by putting the following into the Apache configuration file:

```
LogFormat \
    "%h\" \"%Y%m%d%H%M%S}t,%>s,\"%b\", \"%{Content-Type}o\", \
    \"%U\", \"%{Referer}i\", \"%{User-Agent}i\""
```

To load a log file in that format into MySQL, you can use a statement something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl\_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

The named table should be created to have columns that correspond to those that the [LogFormat](#) line writes to the log file.

Chapter 4. Using MySQL Programs

This chapter provides a brief overview of the command-line programs provided by MySQL AB and discusses the general syntax for specifying options when you run these programs. Most programs have options that are specific to their own operation, but the option syntax is similar for all of them. Later chapters provide more detailed descriptions of individual programs, including which options they recognize.

MySQL AB also provides three GUI client programs for use with MySQL Server:

- **MySQL Administrator:** This tool is used for administering MySQL servers, databases, tables, and user accounts.
- **MySQL Query Browser:** This graphical tool is provided by MySQL AB for creating, executing, and optimizing queries on MySQL databases.
- **MySQL Migration Toolkit:** This tool helps you migrate schemas and data from other relational database management systems for use with MySQL.

These GUI programs each have their own manuals that you can access at <http://dev.mysql.com/doc/>.

4.1. Overview of MySQL Programs

MySQL AB provides several types of programs:

- The MySQL server and server startup scripts:
 - `mysqld` is the MySQL server.
 - `mysqld_safe`, `mysql.server`, and `mysqld_multi` are server startup scripts.
 - `mysql_install_db` initializes the data directory and the initial databases.
 - MySQL Instance Manager monitors and manages MySQL Server instances.

[Chapter 5, Database Administration](#), discusses these programs further

- Client programs that access the server:
 - `mysql` is a command-line client for executing SQL statements interactively or in batch mode.
 - `mysqladmin` is an administrative client.
 - `mysqlcheck` performs table maintenance operations.
 - `mysqldump` and `mysqlhotcopy` make database backups.
 - `mysqlimport` imports data files.
 - `mysqlshow` displays information about databases and tables.

[Chapter 7, Client and Utility Programs](#), discusses these programs further

- Utility programs that operate independently of the server:
 - `myisamchk` performs table maintenance operations.
 - `myisampack` produces compressed, read-only tables.
 - `mysqlbinlog` is a tool for processing binary log files.
 - `pererror` displays the meaning of error codes.

[Chapter 7, Client and Utility Programs](#), discusses these programs further

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see [Chapter 2, *Installing and Upgrading MySQL*](#), for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install something else.

4.2. Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. “`shell>`” represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are `$` for `sh` or `bash`, `%` for `csh` or `tcsh`, and `C:\>` for the Windows `command.com` or `cmd.exe` command interpreters.

```
shell> mysql -u root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump --user=root personnel
```

Arguments that begin with a single or double dash (`-`, `--`) are option arguments. Options typically specify the type of connection a program should make to the server or affect its operational mode. Option syntax is described in [Section 4.3, “Specifying Program Options”](#).

Non-option arguments (arguments with no leading dash) provide additional information to the program. For example, the `mysql` program interprets the first non-option argument as a database name, so the command `mysql -u root test` indicates that you want to use the `test` database.

Later sections that describe individual programs indicate which options a program understands and describe the meaning of any additional non-option arguments.

Some options are common to a number of programs. The most common of these are the `--host` (or `-h`), `--user` (or `-u`), and `--password` (or `-p`) options that specify connection parameters. They indicate the host where the MySQL server is running, and the username and password of your MySQL account. All MySQL client programs understand these options; they allow you to specify which server to connect to and the account to use on that server.

Other connection options are `--port` (or `-P`) to specify a TCP/IP port number and `--socket` (or `-S`) to specify a Unix socket file on Unix (or named pipe name on Windows).

The default hostname is `localhost`. For client programs on Unix, the hostname `localhost` is special. It causes the client to connect to the MySQL server through a Unix socket file. This occurs even if a `--port` or `-P` option is given to specify a port number. To ensure that the client makes a TCP/IP connection to the local server, use `--host` or `-h` to specify a hostname value of `127.0.0.1`, or the IP address or name of the local server. You can also specify the connection protocol explicitly, even for `localhost`, by using the `--protocol=tcp` option.

On Windows, the hostname `.` causes the client to connect to the local server using a named pipe, if the server has named-pipe connections enabled. If named-pipe connections are not enabled, an error occurs.

You may find it necessary to invoke MySQL programs using the pathname to the `bin` directory in which they are installed. This is likely to be the case if you get a “program not found” error whenever you attempt to run a MySQL program from any directory other than the `bin` directory. To make it more convenient to use MySQL, you can add the pathname of the `bin` directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire pathname. For example, if `mysql` is installed in `/usr/local/mysql/bin`, you’ll be able to run it by invoking it as `mysql`; it will not be necessary to invoke it as `/usr/local/mysql/bin/mysql`.

Consult the documentation for your command interpreter for instructions on setting your `PATH` variable. The syntax for setting environment variables is interpreter-specific. (Some information is given in [Section 4.4, “Setting Environment Variables”](#).) After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.3. Specifying Program Options

There are several ways to specify options for MySQL programs:

- List the options on the command line following the program name. This is most common for options that apply to a specific invoca-

tion of the program.

- List the options in an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.
- List the options in environment variables (see [xref linkend="setting-environment-variables"/>](#)). This method is useful for options that you want to apply each time the program runs. In practice, option files are used more commonly for this purpose. However, [Section 5.12.2, “Running Multiple Servers on Unix”](#), discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for both the server and client programs.

MySQL programs determine which options are given first by examining environment variables, then by reading option files, and then by checking the command line. This means that environment variables have the lowest precedence and command-line options the highest.

Because options are processed in order, if an option is specified multiple times, the last occurrence takes precedence. The following command causes `mysql` to connect to the server running on `localhost`:

```
shell> mysql -h example.com -h localhost
```

If conflicting or related options are given, later options take precedence over earlier options. The following command runs `mysql` in “no column names” mode:

```
shell> mysql --column-names --skip-column-names
```

An option can be specified by writing it in full or as any unambiguous prefix. For example, the `--compress` option can be given to `mysqldump` as `--compr`, but not as `--comp` because that is ambiguous:

```
shell> mysqldump --comp
mysqldump: ambiguous option '--comp' (compatible, compress)
```

Be aware that the use of option prefixes can cause problems in the event that new options are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

You can take advantage of the way that MySQL programs process options by specifying default values for a program's options in an option file. That enables you to avoid typing them each time you run the program, but also allows you to override the defaults if necessary by using command-line options.

4.3.1. Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.
- An option argument begins with one dash or two dashes, depending on whether it has a short name or a long name. Many options have both forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display its help message.
- Option names are case sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)
- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.
- For a long option that takes a value, separate the option name and the value by an ‘=’ sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between: `-hlocalhost` and `-h localhost` are equivalent. An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `-password=pass_val` or as `--password`. In the latter case (with no password value given), the program prompts you for the password. The password option also may be given in short form as `-ppass_val` or as `-p`. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*. The reason for this is that if a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind

of argument. Consequently, the following two commands have two completely different meanings:

```
shell> mysql -ptest
shell> mysql -p test
```

The first command instructs `mysql` to use a password value of `test`, but specifies no default database. The second instructs `mysql` to prompt for the password value and to use `test` as the default database.

Some options control behavior that can be turned on or off. For example, the `mysql` client supports a `--column-names` option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you may want to disable it in some instances, such as when sending the output of `mysql` into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The “enabled” form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

If an option is prefixed by `--loose`, a program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine and list options in an option file. An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that recognize the option process it normally, and versions that do not recognize it issue a warning and ignore it.

Another option that may occasionally be useful with `mysql` is the `--execute` or `-e` option, which can be used to pass SQL statements to the server. When this option is used, `mysql` executes the statements and exits. The statements must be enclosed by quotation marks. For example, you can use the following command to obtain a list of user accounts:

```
shell> mysql -u root -p --execute="SELECT User, Host FROM user" mysql
Enter password: *****
+-----+-----+
| User | Host |
+-----+-----+
| root | gigan |
| root | gigan |
|      | localhost |
| jon  | localhost |
| root | localhost |
+-----+-----+
shell>
```

Note that the long form (`--execute`) is followed by an equals sign (`=`).

If you wish to use quoted values within a statement, you will either need to escape the inner quotes, or use a different type of quotes within the statement from those used to quote the statement itself. The capabilities of your command processor dictate your choices for whether you can use single or double quotation marks and the syntax for escaping quote characters. For example, if your command processor supports quoting with single or double quotes, you can double quotes around the statement, and single quotes for any quoted values within the statement.

In the preceding example, the name of the `mysql` database was passed as a separate argument. However, the same statement could have been executed using this command, which specifies no default database:

```
mysql> mysql -u root -p --execute="SELECT User, Host FROM mysql.user"
```

Multiple SQL statements may be passed on the command line, separated by semicolons:

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW();"
Enter password: *****
+-----+
| VERSION() |
+-----+
| 5.1.5-alpha-log |
+-----+
+-----+
| NOW() |
+-----+
| 2006-01-05 21:19:04 |
+-----+
```

The `--execute` or `-e` option may also be used to pass commands in an analogous fashion to the `ndb_mgm` management client for MySQL Cluster. See [Section 16.3.6, “Safe Shutdown and Restart”](#), for an example.

4.3.2. Using Option Files

Most MySQL programs can read startup options from option files (also sometimes called configuration files). Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program. For the MySQL server, MySQL provides a number of [preconfigured option files](#).

To determine whether a program reads option files, invoke it with the `--help` option (`--verbose` and `--help` for `mysqld`). If the program reads option files, the help message indicates which files it looks for and which option groups it recognizes.

Note

Option files used with MySQL Cluster programs are covered in [Section 16.4, “MySQL Cluster Configuration”](#).

On Windows, MySQL programs read startup options from the following files:

Filename	Purpose
<code>WINDIR\my.ini</code>	Global options
<code>C:\my.cnf</code>	Global options
<code>INSTALLDIR\my.ini</code>	Global Options
<code>defaults-extra-file</code>	The file specified with <code>--defaults-extra-file=path</code> , if any

`WINDIR` represents the location of your Windows directory. This is commonly `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
C:\> echo %WINDIR%
```

`INSTALLDIR` represents the installation directory of MySQL. This is typically `C:\PROGRAMDIR\MySQL\MySQL 5.1 Server` where `PROGRAMDIR` represents the programs directory (usually [Program Files](#) on English-language versions of Windows), when MySQL 5.1 has been installed using the installation and configuration wizards. See [Section 2.3.4.14, “The Location of the my.ini File”](#).

On Unix, MySQL programs read startup options from the following files:

Filename	Purpose
<code>/etc/my.cnf</code>	Global options
<code>/etc/mysql/my.cnf</code>	Global options (as of MySQL 5.1.15)
<code>\$MYSQL_HOME/my.cnf</code>	Server-specific options
<code>defaults-extra-file</code>	The file specified with <code>--defaults-extra-file=path</code> , if any
<code>~/.my.cnf</code>	User-specific options

`MYSQL_HOME` is an environment variable containing the path to the directory in which the server-specific `my.cnf` file resides.

If `MYSQL_HOME` is not set and you start the server using the `mysqld_safe` program, `mysqld_safe` attempts to set `MYSQL_HOME`

as follows:

- Let *BASEDIR* and *DATADIR* represent the pathnames of the MySQL base directory and data directory, respectively.
- If there is a *my.cnf* file in *DATADIR* but not in *BASEDIR*, *mysqld_safe* sets *MYSQL_HOME* to *DATADIR*.
- Otherwise, if *MYSQL_HOME* is not set and there is no *my.cnf* file in *DATADIR*, *mysqld_safe* sets *MYSQL_HOME* to *BASEDIR*.

In MySQL 5.1, use of *DATADIR* as the location for *my.cnf* is deprecated. *BASEDIR* is a better location.

Typically, *DATADIR* is */usr/local/mysql/data* for a binary installation or */usr/local/var* for a source installation. Note that this is the data directory location that was specified at configuration time, not the one specified with the *--datadir* option when *mysqld* starts. Use of *--datadir* at runtime has no effect on where the server looks for option files, because it looks for them before processing any options.

MySQL looks for option files in the order just described and reads any that exist. If an option file that you want to use does not exist, create it with a plain text editor.

If multiple instances of a given option are found, the last instance takes precedence. There is one exception: For *mysqld*, the *first* instance of the *--user* option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

Note

On Unix platforms, MySQL ignores configuration files that are world-writable. This is intentional, and acts as a security measure.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the *--help* option.

The syntax for specifying options in an option file is similar to command-line syntax, except that you omit the leading two dashes. For example, *--quick* or *--host=localhost* on the command line should be specified as *quick* or *host=localhost* in an option file. To specify an option of the form *--loose-opt_name* in an option file, write it as *loose-opt_name*.

Empty lines in option files are ignored. Non-empty lines can take any of the following forms:

- *#comment*, *;comment*

Comment lines start with *#* or *;*. A *#* comment can start in the middle of a line as well.

- *[group]*

group is the name of the program or group for which you want to set options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given.

- *opt_name*

This is equivalent to *--opt_name* on the command line.

- *opt_name=value*

This is equivalent to *--opt_name=value* on the command line. In an option file, you can have spaces around the *=* character, something that is not true on the command line. You can enclose the value within single quotes or double quotes, which is useful if the value contains a *#* comment character or whitespace.

For options that take a numeric value, the value can be given with a suffix of *K*, *M*, or *G* (either uppercase or lowercase) to indicate a multiplier of 1024, 1024² or 1024³. For example, the following command tells *mysqladmin* to ping the server 1024 times, sleeping 10 seconds between each ping:

```
mysql> mysqladmin --count=1K --sleep=10 ping
```

Leading and trailing blanks are automatically deleted from option names and values. You may use the escape sequences *\b*, *\t*, *\n*, *\r*, **, and *\s* in option values to represent the backspace, tab, newline, carriage return, backslash, and space characters.

Because the ‘\\’ escape sequence represents a single backslash, you must write each ‘\’ as ‘\\’. Alternatively, you can specify the value using ‘/’ rather than ‘\’ as the pathname separator.

If an option group name is the same as a program name, options in the group apply specifically to that program. For example, the `[mysqld]` and `[mysql]` groups apply to the `mysqld` server and the `mysql` client program, respectively.

The `[client]` option group is read by all client programs (but *not* by `mysqld`). This allows you to specify options that apply to all clients. For example, `[client]` is the perfect group to use to specify the password that you use to connect to the server. (But make sure that the option file is readable and writable only by yourself, so that other people cannot find out your password.) Be sure not to put an option in the `[client]` group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

The preceding option file uses `var_name=value` syntax for the lines that set the `key_buffer_size` and `max_allowed_packet` variables.

Here is a typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password="my_password"

[mysql]
no-auto-rehash
connect_timeout=2

[mysqlhotcopy]
interactive-timeout
```

If you want to create option groups that should be read by `mysqld` servers from a specific MySQL release series only, you can do this by using groups with names of `[mysqld-5.0]`, `[mysqld-5.1]`, and so forth. The following group indicates that the `--new` option should be used only by MySQL servers with 5.1.x version numbers:

```
[mysqld-5.1]
new
```

It is possible to use `!include` directives in option files to include other option files and `!includedir` to search specific directories for option files. For example, to include the `/home/mydir/myopt.cnf` file, use the following directive:

```
!include /home/mydir/myopt.cnf
```

To search the `/home/mydir` directory and read option files found there, use this directive:

```
!includedir /home/mydir
```

There is no guarantee about the order in which the option files in the directory will be read.

Note

Currently, any files to be found and included using the `!includedir` directive on Unix operating systems *must* have filenames ending in `.cnf`. On Windows, this directive checks for files with the `.ini` or `.cnf` extension.

Write the contents of an included option file like any other option file. That is, it should contain groups of options, each preceded by a `[group]` line that indicates the program to which the options apply.

While an included file is being processed, only those options in groups that the current program is looking for are used. Other groups are ignored. Suppose that a `my.cnf` file contains this line:

```
!include /home/mydir/myopt.cnf
```

And suppose that `/home/mydir/myopt.cnf` looks like this:

```
[mysqladmin]
force

[mysqld]
key_buffer_size=16M
```

If `my.cnf` is processed by `mysqld`, only the `[mysqld]` group in `/home/mydir/myopt.cnf` is used. If the file is processed by `mysqladmin`, only the `[mysqladmin]` group is used. If the file is processed by any other program, no options in `/home/mydir/myopt.cnf` are used.

The `!includedir` directive is processed similarly except that all option files in the named directory are read.

If you have a source distribution, you can find sample option files named `my-xxx.cnf` in the `support-files` directory. If you have a binary distribution, look in the `support-files` directory under your MySQL installation directory. On Windows, the sample option files may be located in the MySQL installation directory (see earlier in this section or [Chapter 2, Installing and Upgrading MySQL](#), if you do not know where this is). Currently, there are sample option files for small, medium, large, and very large systems. To experiment with one of these files, copy it to `C:\my.cnf` on Windows or to `.my.cnf` in your home directory on Unix.

Note

On Windows, the `.cnf` or `.ini` option file extension might not be displayed.

Most MySQL programs that support option files handle the following options. They affect option-file handling, so they must be given on the command line and not in an option file. To work properly, each of these options must immediately follow the command name, with the exception that `--print-defaults` may be used immediately after `--defaults-file` or `--defaults-extra-file`. Also, you should avoid the use of the `~` shell metacharacter when specifying filenames because it might not be interpreted as you expect.

- `--no-defaults`

Don't read any option files.

- `--print-defaults`

Print the program name and all options that it gets from option files.

- `--defaults-file=file_name`

Use only the given option file. `file_name` is the full pathname to the file. If the file does not exist or is otherwise inaccessible, the program will exit with an error.

- `--defaults-extra-file=file_name`

Read this option file after the global option file but (on Unix) before the user option file. `file_name` is the full pathname to the file. If the file does not exist or is otherwise inaccessible, the program will exit with an error.

- `--defaults-group-suffix=str`

If this option is given, the program reads not only its usual option groups, but also groups with the usual names and a suffix of `str`. For example, the `mysql` client normally reads the `[client]` and `[mysql]` groups. If the `--default-group-suffix=_other` option is given, `mysql` also reads the `[client_other]` and `[mysql_other]` groups.

In shell scripts, you can use the `my_print_defaults` program to parse option files and see what options would be used by a given program. The following example shows the output that `my_print_defaults` might produce when asked to show the options found in the `[client]` and `[mysql]` groups:

```
shell> my_print_defaults client mysql
```

```
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Note for developers: Option file handling is implemented in the C client library simply by processing all options in the appropriate group or groups before any command-line arguments. This works well for programs that use the last instance of an option that is specified multiple times. If you have a C or C++ program that handles multiply specified options this way but that doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard MySQL clients to see how to do this.

Several other language interfaces to MySQL are based on the C client library, and some of them provide a way to access option file contents. These include Perl and Python. For details, see the documentation for your preferred interface.

4.3.2.1. Preconfigured Option Files

MySQL provides a number of preconfigured option files that can be used as a basis for tuning the MySQL server. Look in your installation directory for files such as `my-small.cnf`, `my-medium.cnf`, `my-large.cnf`, and `my-huge.cnf`, which you can rename and copy to the appropriate location for use as a base configuration file. Regarding names and appropriate location, see the general information provided in [Section 4.3.2, “Using Option Files”](#). On Windows, those files have a `.ini` rather than a `.cnf` extension.

4.3.3. Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime. Program variables are set the same way as any other long option that takes a value. For example, `mysql` has a `max_allowed_packet` variable that controls the maximum size of its communication buffer. To set the `max_allowed_packet` variable for `mysql` to a value of 16MB, use either of the following commands:

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. For variables that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024 , 1024^2 or 1024^3 . (For example, when used to set `max_allowed_packet`, the suffixes indicate units of kilobytes, megabytes, or gigabytes.)

In an option file, variable settings are given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in a variable name can be specified as dashes. The following option groups are equivalent. Both set the size of the server's key buffer to 512MB:

```
[mysqld]
key_buffer_size=512M

[mysqld]
key-buffer-size=512M
```

Note

Before MySQL 4.0.2, the only syntax for setting program variables was `--set-variable=option=value` (or `set-variable=option=value` in option files). This syntax still is recognized, but is deprecated as of MySQL 4.0.2.

Many server system variables can also be set at runtime. For details, see [Section 5.2.4.2, “Dynamic System Variables”](#).

4.4. Setting Environment Variables

Environment variables can be set at the command prompt to affect the current invocation of your command processor, or set permanently to affect future invocations. To set a variable permanently, you can set it in a startup file or by using the interface provided by your system for this purpose. Consult the documentation for your command interpreter for specific details. [Section 2.14, “Environment Variables”](#), lists all environment variables that affect MySQL program operation.

To specify a value for an environment variable, use the syntax appropriate for your command processor. For example, on Windows or NetWare, you can set the `USER` variable to specify your MySQL account name. To do so, use this syntax:

```
SET USER=your_name
```

The syntax on Unix depends on your shell. Suppose that you want to specify the TCP/IP port number using the `MYSQL_TCP_PORT` variable. Typical syntax (such as for `sh`, `bash`, `zsh`, and so on) is as follows:

```
MYSQL_TCP_PORT=3306  
export MYSQL_TCP_PORT
```

The first command sets the variable, and the `export` command exports the variable to the shell environment so that its value becomes accessible to MySQL and other processes.

For `csh` and `tcsh`, use `setenv` to make the shell variable available to the environment:

```
setenv MYSQL_TCP_PORT 3306
```

The commands to set environment variables can be executed at your command prompt to take effect immediately, but the settings persist only until you log out. To have the settings take effect each time you log in, use the interface provided by your system or place the appropriate command or commands in a startup file that your command interpreter reads each time it starts.

On Windows, you can set environment variables using the System Control Panel (under Advanced).

On Unix, typical shell startup files are `.bashrc` or `.bash_profile` for `bash`, or `.tcshrc` for `tcsh`.

Suppose that your MySQL programs are installed in `/usr/local/mysql/bin` and that you want to make it easy to invoke these programs. To do this, set the value of the `PATH` environment variable to include that directory. For example, if your shell is `bash`, add the following line to your `.bashrc` file:

```
PATH=${PATH}:/usr/local/mysql/bin
```

`bash` uses different startup files for login and non-login shells, so you might want to add the setting to `.bashrc` for login shells and to `.bash_profile` for non-login shells to make sure that `PATH` is set regardless.

If your shell is `tcsh`, add the following line to your `.tcshrc` file:

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

If the appropriate startup file does not exist in your home directory, create it with a text editor.

After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

Chapter 5. Database Administration

This chapter covers topics that deal with administering a MySQL installation:

- Configuring the server
- Managing user accounts
- Performing backups
- The server log files

5.1. Overview of Server-Side Programs

The MySQL server, `mysqld`, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that perform setup operations when you install MySQL or that assist you in starting and stopping the server. This section provides an overview of the server and related programs. The following sections provide more detailed information about each of these programs.

Each MySQL program takes many different options. Most programs provide a `--help` option that you can use to get a description of the program's different options. For example, try `mysqld --help`.

You can override default option values for MySQL programs by specifying options on the command line or in an option file. [Section 4.3, “Specifying Program Options”](#).

The following list briefly describes the MySQL server and server-related programs:

- `mysqld`

The SQL daemon (that is, the MySQL server). To use client programs, `mysqld` must be running, because clients gain access to databases by connecting to the server. See [Section 5.2, “mysqld — The MySQL Server”](#).

- `mysqld_safe`

A server startup script. `mysqld_safe` attempts to start `mysqld`. See [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#).

- `mysql.server`

A server startup script. This script is used on systems that use System V-style run directories containing scripts that start system services for particular run levels. It invokes `mysqld_safe` to start the MySQL server. See [Section 5.3.2, “mysql.server — MySQL Server Startup Script”](#).

- `mysqld_multi`

A server startup script that can start or stop multiple servers installed on the system. See [Section 5.3.3, “mysqld_multi — Manage Multiple MySQL Servers”](#). An alternative to `mysqld_multi` is `mysqlmanager`, the MySQL Instance Manager. See [Section 5.4, “mysqlmanager — The MySQL Instance Manager”](#).

- `mysqlmanager`

The MySQL Instance Manager, a program for monitoring and managing MySQL servers. See [Section 5.4, “mysqlmanager — The MySQL Instance Manager”](#).

There are several other programs that are related to MySQL installation or upgrading:

- `comp_err`

This program is used during the MySQL build/installation process. It compiles error message files from the error source files. See [Section 5.5.1, “comp_err — Compile MySQL Error Message File”](#).

- `make_binary_distribution`

This program makes a binary release of a compiled MySQL. This could be sent by FTP to [/pub/mysql/upload/](ftp://pub.mysql.com) on <ftp.mysql.com> for the convenience of other MySQL users.

- `make_win_bin_dist`

This program is used on Windows. It packages a MySQL distribution for installation after the source distribution has been built. See [Section 5.5.2, “make_win_bin_dist — Package MySQL Distribution as ZIP Archive”](#).

- `mysql_fix_privilege_tables`

This program is used after a MySQL upgrade operation. It updates the grant tables with any changes that have been made in newer versions of MySQL. See [Section 5.5.3, “mysql_fix_privilege_tables — Upgrade MySQL System Tables”](#).

Note: As of MySQL 5.1.7, this program has been superseded by `mysql_upgrade`.

- `mysql_install_db`

This script creates the MySQL database and initializes the grant tables with default privileges. It is usually executed only once, when first installing MySQL on a system. See [Section 2.10.2, “Unix Post-Installation Procedures”](#), and [Section 5.5.4, “mysql_install_db — Initialize MySQL Data Directory”](#).

- `mysql_secure_installation`

This program enables you to improve the security of your MySQL installation. SQL. See [Section 5.5.5, “mysql_secure_installation — Improve MySQL Installation Security”](#).

- `mysql_tzinfo_to_sql`

This program loads the time zone tables in the `mysql` database using the contents of the host system *zoneinfo* database (the set of files describing time zones). SQL. See [Section 5.5.6, “mysql_tzinfo_to_sql — Load the Time Zone Tables”](#).

- `mysql_upgrade`

This program is used after a MySQL upgrade operation. It checks tables for incompatibilities and repairs them if necessary, and updates the grant tables with any changes that have been made in newer versions of MySQL. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

5.2. `mysqld` — The MySQL Server

`mysqld` is the MySQL server. The following discussion covers these MySQL server configuration topics:

- Startup options that the server supports
- Server system variables
- Server status variables
- How to set the server SQL mode
- The server shutdown process

Note

Not all storage engines are supported by all MySQL server binaries and configurations. To find out how to determine which storage engines are supported by your MySQL server installation, see [Section 12.5.4.14, “SHOW ENGINES Syntax”](#).

5.2.1. Option and Variable Reference

The following table provides a list of all the command line options, server and status variables applicable within `mysqld`.

The table lists command line options (Cmd-line), options valid in configuration files (Option file), server system variables (Server Var), and status variables (Status var) in one unified list, with notification of where each option/variable is valid. If a server option set on the command line or in an option file differs from the name of the corresponding server system or status variable, the variable name is noted immediately below the corresponding option. For status variables, the scope of the variable is shown (Scope) as either global, session, or both. Please see the corresponding sections for details on setting and using the options and variables. Where appropriate, a direct link to further information on the item as available.

Note

This table is part of an ongoing process to expand and simplify the information provided on these elements. Further improvements to the table, and corresponding descriptions will be applied over the coming months.

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
abort-slave-event-count	Y	Y				
Aborted_clients				Y	global	no
Aborted_connects				Y	global	no
allow-suspicious-udfs	Y	Y				
ansi	Y	Y				
auto-increment-increment	Y	Y			both	no
- Variable: auto_increment_increment			Y		both	no
auto-increment-offset	Y	Y			both	no
- Variable: auto_increment_offset			Y		both	no
autocommit	Y	Y	Y		session	yes
automatic_sp_privileges	Y	Y	Y		both	no
back_log	Y	Y	Y		both	no
basedir	Y	Y	Y		both	no
bdb-data-direct			Y		both	no
bdb-home	Y	Y			both	no
- Variable: bdb_home			Y		both	no
bdb-lock-detect	Y	Y	Y		both	no
bdb-log-direct			Y		both	no
bdb-logdir	Y	Y			both	no
- Variable: bdb_logdir			Y		both	no
bdb-no-recover	Y	Y				
bdb-no-sync	Y	Y				
bdb-shared-data	Y	Y			both	no
- Variable: bdb_shared_data			Y		both	no
bdb-tmpdir	Y	Y			both	no
- Variable: bdb_tmpdir			Y		both	no
bdb_cache_size	Y	Y	Y		both	no
bdb_log_buffer_size	Y	Y	Y		both	no
bdb_max_lock	Y	Y	Y		both	no
big-tables	Y	Y			both	yes
- Variable: big_tables			Y		both	yes
bind-address	Y	Y				
binlog-do-db	Y	Y				
binlog-ignore-db	Y	Y				
binlog-row-event-max-size	Y	Y				
Binlog_cache_disk_use				Y	global	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
binlog_cache_size	Y	Y	Y		both	yes
Binlog_cache_use				Y	global	no
binlog_format	Y	Y	Y		both	no
bootstrap	Y	Y				
bulk_insert_buffer_size	Y	Y	Y		both	yes
Bytes_received				Y	both	no
Bytes_sent				Y	both	no
character-set-client-handshake	Y					
character-set-filesystem	Y	Y			both	no
- <i>Variable:</i> character_set_filesystem			Y		both	no
character-set-server	Y	Y			both	yes
- <i>Variable:</i> character_set_server			Y		both	yes
character-sets-dir	Y	Y			both	no
- <i>Variable:</i> character_sets_dir			Y		both	no
character_set	Y		Y			no
character_set_client			Y		both	yes
character_set_connection			Y		both	yes
character_set_database			Y		both	yes
character_set_results			Y		both	yes
character_set_system			Y		both	yes
chroot	Y	Y				
collation_connection	Y	Y	Y		both	yes
collation_database	Y	Y	Y		both	yes
collation_server	Y	Y	Y		both	yes
Com_admin_commands				Y	both	no
Com_alter_db				Y	both	no
Com_alter_event				Y	both	no
Com_alter_table				Y	both	no
Com_analyze				Y	both	no
Com_backup				Y	both	no
Com_backup_table				Y	both	no
Com_begin				Y	both	no
Com_call_procedure				Y	both	no
Com_change_db				Y	both	no
Com_change_master				Y	both	no
Com_check				Y	both	no
Com_checksum				Y	both	no
Com_commit				Y	both	no
Com_create_db				Y	both	no
Com_create_event				Y	both	no
Com_create_function				Y	both	no
Com_create_index				Y	both	no
Com_create_table				Y	both	no
Com_create_user				Y	both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_dealloc_sql				Y	both	no
Com_delete				Y	both	no
Com_delete_multi				Y	both	no
Com_do				Y	both	no
Com_drop_db				Y	both	no
Com_drop_event				Y	both	no
Com_drop_function				Y	both	no
Com_drop_index				Y	both	no
Com_drop_table				Y	both	no
Com_drop_user				Y	both	no
Com_execute_sql				Y	both	no
Com_flush				Y	both	no
Com_grant				Y	both	no
Com_ha_close				Y	both	no
Com_ha_open				Y	both	no
Com_ha_read				Y	both	no
Com_help				Y	both	no
Com_insert				Y	both	no
Com_insert_select				Y	both	no
Com_kill				Y	both	no
Com_load				Y	both	no
Com_load_master_data				Y	both	no
Com_load_master_table				Y	both	no
Com_lock_tables				Y	both	no
Com_optimize				Y	both	no
Com_preload_keys				Y	both	no
Com_prepare_sql				Y	both	no
Com_purge				Y	both	no
Com_purge_before_date				Y	both	no
Com_rename_table				Y	both	no
Com_repair				Y	both	no
Com_replace				Y	both	no
Com_replace_select				Y	both	no
Com_reset				Y	both	no
Com_restore				Y	both	no
Com_restore_table				Y	both	no
Com_revoke				Y	both	no
Com_revoke_all				Y	both	no
Com_rollback				Y	both	no
Com_savepoint				Y	both	no
Com_select				Y	both	no
Com_set_option				Y	both	no
Com_show_binlog_events				Y	both	no
Com_show_binlogs				Y	both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_show_charsets				Y	both	no
Com_show_collations				Y	both	no
Com_show_column_types				Y	both	no
Com_show_create_db				Y	both	no
Com_show_create_event				Y	both	no
Com_show_create_table				Y	both	no
Com_show_databases				Y	both	no
Com_show_engine_logs				Y	both	no
Com_show_engine_mutex				Y	both	no
Com_show_engine_status				Y	both	no
Com_show_errors				Y	both	no
Com_show_events				Y	both	no
Com_show_fields				Y	both	no
Com_show_grants				Y	both	no
Com_show_keys				Y	both	no
Com_show_master_status				Y	both	no
Com_show_new_master				Y	both	no
Com_show_open_tables				Y	both	no
Com_show_plugins				Y	both	no
Com_show_privileges				Y	both	no
Com_show_processlist				Y	both	no
Com_show_slave_hosts				Y	both	no
Com_show_slave_status				Y	both	no
Com_show_status				Y	both	no
Com_show_storage_engines				Y	both	no
Com_show_tables				Y	both	no
Com_show_triggers				Y	both	no
Com_show_variables				Y	both	no
Com_show_warnings				Y	both	no
Com_slave_start				Y	both	no
Com_slave_stop				Y	both	no
Com_stmt_close				Y	both	no
Com_stmt_execute				Y	both	no
Com_stmt_fetch				Y	both	no
Com_stmt_prepare				Y	both	no
Com_stmt_reset				Y	both	no
Com_stmt_send_long_data				Y	both	no
Com_truncate				Y	both	no
Com_unlock_tables				Y	both	no
Com_update				Y	both	no
Com_update_multi				Y	both	no
Com_xa_commit				Y	both	no
Com_xa_end				Y	both	no
Com_xa_prepare				Y	both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_xa_recover				Y	both	no
Com_xa_rollback				Y	both	no
Com_xa_start				Y	both	no
completion_type	Y	Y	Y		both	yes
Compression				Y	session	no
concurrent-insert	Y	Y			both	yes
- Variable: concurrent_insert			Y		both	yes
connect_timeout	Y	Y	Y		both	yes
Connections				Y	global	no
console	Y	Y				
core-file	Y	Y				
crash_binlog_innodb	Y	Y				
Created_tmp_disk_tables				Y	both	no
Created_tmp_files				Y	global	no
Created_tmp_tables				Y	both	no
datadir	Y	Y	Y		both	no
date_format	Y	Y	Y		both	yes
datetime_format	Y	Y	Y		both	yes
debug	Y	Y				
default-character-set	Y	Y				
default-collation	Y		Y			no
default-storage-engine	Y	Y				
default-table-type	Y	Y				
default-time-zone	Y	Y				
default_week_format	Y	Y	Y		both	yes
defaults-extra-file	Y					
defaults-file	Y					
defaults-group-suffix	Y					
delay-key-write	Y	Y			both	yes
- Variable: delay_key_write			Y		both	yes
delay-key-write-for-all-tables	Y					
Delayed_errors				Y	global	no
delayed_insert_limit	Y	Y	Y		both	yes
Delayed_insert_threads				Y	global	no
delayed_insert_timeout	Y	Y	Y		both	yes
delayed_queue_size	Y	Y	Y		both	yes
Delayed_writes				Y	global	no
des-key-file	Y	Y	Y		both	no
disconnect-slave-event-count	Y	Y				
div_precision_increment	Y	Y	Y		both	yes
enable-locking	Y					
enable-pstack	Y	Y				
engine-condition-pushdown	Y	Y			both	yes
- Variable: engine_condition_pushdown			Y		both	yes

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
error_count			Y		session	no
event-scheduler	Y	Y				no
- Variable: event_scheduler			Y			no
exit-info	Y	Y				
expire_logs_days	Y	Y	Y		global	yes
external-locking	Y	Y				
flush	Y	Y	Y		both	yes
Flush_commands				Y	global	no
flush_time	Y	Y	Y		both	yes
foreign_key_checks	Y		Y		session	yes
ft_boolean_syntax	Y	Y	Y		both	yes
ft_max_word_len	Y	Y	Y		both	yes
ft_min_word_len	Y	Y	Y		global	yes
ft_query_expansion_limit	Y	Y	Y		both	yes
ft_stopword_file	Y	Y	Y		both	yes
gdb	Y	Y				
general-log	Y	Y			global	no
- Variable: general_log			Y		global	no
general_log_file			Y		global	no
group_concat_max_len	Y	Y	Y		both	yes
Handler_commit				Y	both	no
Handler_delete				Y	both	no
Handler_discover				Y	both	no
Handler_prepare				Y	both	no
Handler_read_first				Y	both	no
Handler_read_key				Y	both	no
Handler_read_next				Y	both	no
Handler_read_prev				Y	both	no
Handler_read_rnd				Y	both	no
Handler_read_rnd_next				Y	both	no
Handler_rollback				Y	both	no
Handler_savepoint				Y	both	no
Handler_savepoint_rollback				Y	both	no
Handler_update				Y	both	no
Handler_write				Y	both	no
have_archive			Y		global	no
have_bdb			Y		global	no
have_blackhole_engine			Y		global	no
have_compress			Y		global	no
have_crypt			Y		global	no
have_csv			Y		global	no
have_dynamic_loading			Y		global	no
have_example_engine			Y		global	no
have_federated_engine			Y		global	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
have_geometry			Y		global	no
have_innodb			Y		global	no
have_isam			Y		global	no
have_merge_engine			Y		global	no
have_ndbcluster			Y		global	no
have_openssl			Y		global	no
have_partition_engine			Y		global	no
have_partitioning			Y		global	no
have_query_cache			Y		global	no
have_raid			Y		global	no
have_row_based_replication			Y		global	no
have_rtree_keys			Y		global	no
have_symlink			Y		global	no
help	Y					
hostname			Y		global	no
identity	Y	Y	Y		session	yes
init-file	Y	Y			both	no
- Variable: init_file			Y		both	no
init_connect	Y	Y	Y		global	no
init_slave	Y	Y	Y		both	no
innodb	Y	Y				
innodb-safe-binlog	Y	Y				
innodb_additional_mem_pool_size	Y	Y	Y		both	no
innodb_autoextend_increment	Y	Y	Y		both	yes
innodb_autoinc_lock_mode	Y	Y	Y		global	yes
innodb_buffer_pool_ave_mem_mb	Y	Y	Y		both	no
Innodb_buffer_pool_pages_data				Y	both	no
Innodb_buffer_pool_pages_dirty				Y	both	no
Innodb_buffer_pool_pages_flushed				Y	both	no
Innodb_buffer_pool_pages_free				Y	both	no
Innodb_buffer_pool_pages_latched				Y	both	no
Innodb_buffer_pool_pages_misc				Y	both	no
Innodb_buffer_pool_pages_total				Y	both	no
Innodb_buffer_pool_read_ahead_rnd				Y	both	no
Innodb_buffer_pool_read_ahead_seq				Y	both	no
Innodb_buffer_pool_read_requests				Y	both	no
Innodb_buffer_pool_reads				Y	both	no
innodb_buffer_pool_size	Y	Y	Y		both	no
Innodb_buffer_pool_wait_free				Y	both	no
Innodb_buffer_pool_write_requests				Y	both	no
innodb_checksums	Y	Y	Y		both	yes
innodb_commit_concurrency	Y	Y	Y		both	yes
innodb_concurrency_tickets	Y	Y	Y		both	yes
innodb_data_file_path	Y	Y	Y		both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
Innodb_data_fsyncs				Y	both	no
innodb_data_home_dir	Y	Y	Y		both	no
Innodb_data_pending_fsyncs				Y	both	no
Innodb_data_pending_reads				Y	both	no
Innodb_data_pending_writes				Y	both	no
Innodb_data_read				Y	both	no
Innodb_data_reads				Y	both	no
Innodb_data_writes				Y	both	no
Innodb_data_written				Y	both	no
Innodb_dblwr_pages_written				Y	both	no
Innodb_dblwr_writes				Y	both	no
innodb_doublewrite	Y	Y	Y		both	no
innodb_fast_shutdown	Y	Y	Y		both	no
innodb_file_io_threads	Y	Y	Y		both	no
innodb_file_per_table	Y	Y	Y		both	no
innodb_flush_log_at_trx_commit	Y	Y	Y		global	yes
innodb_flush_method	Y	Y	Y		both	no
innodb_force_recovery	Y	Y	Y		both	no
innodb_lock_wait_timeout	Y	Y	Y		both	no
innodb_locks_unsafe_for_binlog	Y	Y	Y		both	no
innodb_log_arch_dir	Y	Y	Y		both	no
innodb_log_archive	Y	Y	Y		both	no
innodb_log_buffer_size	Y	Y	Y		both	no
innodb_log_file_size	Y	Y	Y		both	no
innodb_log_files_in_group	Y	Y	Y		both	no
innodb_log_group_home_dir	Y	Y	Y		both	no
Innodb_log_waits				Y	both	no
Innodb_log_write_requests				Y	both	no
Innodb_log_writes				Y	both	no
innodb_max_dirty_pages_pct	Y	Y	Y		both	yes
innodb_max_purge_lag	Y	Y	Y		both	yes
innodb_mirrored_log_groups	Y	Y	Y		both	no
innodb_open_files	Y	Y	Y		both	no
Innodb_os_log_fsyncs				Y	both	no
Innodb_os_log_pending_fsyncs				Y	both	no
Innodb_os_log_pending_writes				Y	both	no
Innodb_os_log_written				Y	both	no
Innodb_page_size				Y	both	no
Innodb_pages_created				Y	both	no
Innodb_pages_read				Y	both	no
Innodb_pages_written				Y	both	no
innodb_rollback_on_timeout	Y	Y	Y		both	no
Innodb_row_lock_current_waits				Y	both	no
Innodb_row_lock_time				Y	both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
Innodb_row_lock_time_avg				Y	both	no
Innodb_row_lock_time_max				Y	both	no
Innodb_row_lock_waits				Y	both	no
Innodb_rows_deleted				Y	both	no
Innodb_rows_inserted				Y	both	no
Innodb_rows_read				Y	both	no
Innodb_rows_updated				Y	both	no
innodb_stats_on_metadata	Y	Y	Y			no
innodb_status_file	Y	Y	Y		both	no
innodb_support_xa	Y	Y	Y		both	yes
innodb_sync_spin_loops	Y	Y	Y		both	yes
innodb_table_locks	Y	Y	Y		both	yes
innodb_thread_concurrency	Y	Y	Y		global	yes
innodb_thread_sleep_delay	Y	Y	Y		both	yes
insert_id	Y	Y	Y		session	yes
interactive_timeout	Y	Y	Y		both	yes
isam	Y	Y	Y			no
join_buffer_size	Y	Y	Y		both	yes
keep_files_on_create	Y	Y	Y		global	yes
Key_blocks_not_flushed				Y	both	no
Key_blocks_unused				Y	both	no
Key_blocks_used				Y	both	no
key_buffer_size	Y	Y	Y		both	yes
key_cache_age_threshold	Y	Y	Y		both	no
key_cache_block_size	Y	Y	Y		both	no
key_cache_division_limit	Y	Y	Y		both	no
Key_read_requests				Y	both	no
Key_reads				Y	both	no
Key_write_requests				Y	both	no
Key_writes				Y	both	no
language	Y	Y	Y		both	no
large-pages	Y	Y			both	no
- Variable: large_pages			Y		both	no
large_files_support			Y		both	no
last_insert_id	Y	Y	Y		session	yes
Last_query_cost				Y	both	no
lc_time_names			Y		both	yes
license			Y		both	no
local-infile	Y	Y			global	yes
- Variable: local_infile			Y		global	yes
log	Y	Y	Y		both	no
log-bin	Y	Y			both	no
- Variable: log_bin			Y		both	no
log-bin-index	Y	Y	Y		both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
log-bin-trust-function-creators	Y	Y			both	yes
- Variable: log_bin_trust_function_creators			Y		both	yes
log-bin-trust-routine-creators	Y	Y			both	no
- Variable: log_bin_trust_routine_creators			Y		both	no
log-error	Y	Y			both	no
- Variable: log_error			Y		both	no
log-isam	Y	Y	Y		both	no
log-long-format	Y	Y				
log-queries-not-using-indexes	Y	Y			both	yes
- Variable: log_queries_not_using_indexes			Y		both	yes
log-short-format	Y	Y				
log-slave-updates	Y	Y			both	no
- Variable: log_slave_updates			Y		both	no
log-slow-admin-statements	Y	Y				
log-slow-queries	Y	Y			both	no
- Variable: log_slow_queries			Y		both	no
log-tc	Y	Y				
log-tc-size	Y	Y				
log-update	Y					
log-warnings	Y	Y			both	yes
- Variable: log_warnings			Y		both	yes
log_output	Y	Y	Y		global	yes
long_query_time	Y	Y	Y		both	yes
low-priority-updates	Y	Y			both	yes
- Variable: low_priority_updates			Y		both	yes
lower_case_file_system	Y	Y	Y		both	no
lower_case_table_names	Y	Y	Y		both	no
master-connect-retry	Y	Y	Y			no
master-host	Y	Y				
master-info-file	Y	Y				
master-password	Y	Y				
master-port	Y	Y				
master-retry-count	Y	Y				
master-ssl	Y	Y				
master-ssl-ca	Y	Y				
master-ssl-capath	Y	Y				
master-ssl-cert	Y	Y				
master-ssl-cipher	Y	Y				
master-ssl-key	Y	Y				
master-user	Y	Y				
Master_Log_File	Y	Y	Y		both	no
Master_SSL_Allowed	Y	Y	Y		both	no
Master_SSL_CA_File	Y	Y	Y		both	no
Master_SSL_CA_Path	Y	Y	Y		both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
max-binlog-dump-events	Y	Y				
max_allowed_packet	Y	Y	Y		both	yes
max_binlog_cache_size	Y	Y	Y		global	yes
max_binlog_size	Y	Y	Y		global	yes
max_connect_errors	Y	Y	Y		global	yes
max_connections	Y	Y	Y		global	yes
max_delayed_threads	Y	Y	Y		global	yes
max_error_count	Y	Y	Y		both	yes
max_heap_table_size	Y	Y	Y		both	yes
max_insert_delayed_threads			Y		global	yes
max_join_size	Y	Y	Y		both	yes
max_length_for_sort_data	Y	Y	Y		both	yes
max_prepared_stmt_count	Y	Y	Y		global	yes
max_relay_log_size	Y	Y	Y		global	yes
max_seeks_for_key	Y	Y	Y		both	yes
max_sort_length	Y	Y	Y		both	yes
max_sp_recursion_depth	Y	Y	Y		global	yes
max_tmp_tables	Y	Y	Y		both	yes
Max_used_connections				Y	global	no
max_user_connections	Y	Y	Y		both	yes
max_write_lock_count	Y	Y	Y		global	yes
memlock	Y	Y			both	no
- Variable: locked_in_memory			Y		both	no
merge	Y	Y	Y		both	no
multi_range_count	Y	Y	Y		both	yes
multi_read_range	Y	Y	Y		both	yes
myisam-recover	Y	Y	Y		both	no
myisam_block_size	Y	Y	Y		both	yes
myisam_data_pointer_size	Y	Y	Y		global	yes
myisam_max_extra_sort_file_size	Y	Y	Y		both	no
myisam_max_sort_file_size	Y	Y	Y		both	yes
myisam_repair_threads	Y	Y	Y		both	yes
myisam_sort_buffer_size	Y	Y	Y		both	yes
myisam_stats_method	Y	Y	Y		both	yes
myisam_use_mmap	Y	Y	Y		global	no
named_pipe	Y	Y				
ndb_autoincrement_prefetch_sz	Y	Y	Y		both	yes
ndb_cache_check_time	Y	Y	Y		both	yes
ndb_cluster_connection_pool	Y	Y	Y			no
Ndb_cluster_node_id				Y	both	no
Ndb_config_from_host				Y	both	no
Ndb_config_from_port				Y	both	no
ndb_extra_logging	Y	Y	Y		global	yes
ndb_force_send	Y	Y	Y		both	yes

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
ndb_index_stat_cache_entries	Y	Y				
ndb_index_stat_enable	Y	Y				
ndb_index_stat_update_freq	Y	Y				
Ndb_number_of_data_nodes				Y	global	no
ndb_optimized_node_selection	Y	Y				
ndb_report_thresh_binlog_epoch_slip	Y	Y				
ndb_report_thresh_binlog_mem_usage	Y	Y				
ndb_use_copying_alter_table			Y			no
ndb_use_exact_count			Y		both	yes
ndb_use_transactions	Y	Y				
ndb_wait_connected	Y	Y	Y			no
ndbcluster	Y	Y	Y		both	yes
net_buffer_length	Y	Y	Y		both	yes
net_read_timeout	Y	Y	Y		both	yes
net_retry_count	Y	Y	Y		both	yes
net_write_timeout	Y	Y	Y		both	yes
new	Y	Y	Y		both	no
no-defaults	Y					
Not_flushed_delayed_rows				Y	global	no
old	Y	Y	Y			no
old-alter-table	Y	Y				
old-passwords	Y	Y			both	yes
- Variable: old_passwords			Y		both	yes
old-protocol	Y	Y				
old-style-user-limits	Y	Y				
Open_files				Y	global	no
open_files_limit	Y	Y	Y		both	no
Open_streams				Y	global	no
Open_table_definitions				Y	global	no
Open_tables				Y	global	no
Opened_tables				Y	both	no
optimizer_prune_level	Y	Y	Y		both	yes
optimizer_search_depth	Y	Y	Y		both	yes
pid-file	Y	Y	Y		both	no
plugin-innodb	Y	Y				
plugin-innodb-doublewrite	Y	Y			both	no
- Variable: plugin_innodb_doublewrite			Y		both	no
plugin_dir	Y	Y	Y			no
plugin_innodb_additional_mem_pool_size	Y	Y	Y		both	no
plugin_innodb_autoextend_increment	Y	Y	Y		both	yes
plugin_innodb_buffer_pool_ave_mem_mb	Y	Y			both	no
- Variable: innodb_buffer_pool_ave_mem_mb			Y		both	no
plugin_innodb_buffer_pool_size	Y	Y	Y		both	no
plugin_innodb_checksums	Y	Y	Y		both	yes

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
plugin_innodb_commit_concurrency	Y	Y	Y		both	yes
plugin_innodb_concurrency_tickets	Y	Y	Y		both	yes
plugin_innodb_data_file_path	Y	Y	Y		both	no
plugin_innodb_data_home_dir	Y	Y	Y		both	no
plugin_innodb_fast_shutdown	Y	Y	Y		both	no
plugin_innodb_file_io_threads	Y	Y	Y		both	no
plugin_innodb_file_per_table	Y	Y	Y		both	no
plugin_innodb_flush_log_at_trx_commit	Y	Y	Y		global	yes
plugin_innodb_flush_method	Y	Y	Y		both	no
plugin_innodb_force_recovery	Y	Y	Y		both	no
plugin_innodb_lock_wait_timeout	Y	Y	Y		both	no
plugin_innodb_locks_unsafe_for_binlog	Y	Y	Y		both	no
plugin_innodb_log_archive	Y	Y	Y		both	no
plugin_innodb_log_buffer_size	Y	Y	Y		both	no
plugin_innodb_log_file_size	Y	Y	Y		both	no
plugin_innodb_log_files_in_group	Y	Y	Y		both	no
plugin_innodb_log_group_home_dir	Y	Y	Y		both	no
plugin_innodb_max_dirty_pages_pct	Y	Y	Y		both	yes
plugin_innodb_max_purge_lag	Y	Y	Y		both	yes
plugin_innodb_mirrored_log_groups	Y	Y	Y		both	no
plugin_innodb_open_files	Y	Y	Y		both	no
plugin_innodb_rollback_on_timeout	Y	Y	Y			no
plugin_innodb_stats_on_metadata	Y	Y	Y			no
plugin_innodb_status_file	Y	Y	Y			no
plugin_innodb_support_xa	Y	Y	Y		both	yes
plugin_innodb_sync_spin_loops	Y	Y			both	yes
- Variable: innodb_sync_spin_loops			Y		both	yes
plugin_innodb_table_locks	Y	Y	Y		both	yes
plugin_innodb_thread_concurrency	Y	Y	Y		both	yes
plugin_innodb_thread_sleep_delay	Y	Y	Y		both	yes
port	Y	Y	Y		both	no
port-open-timeout	Y	Y				
preload_buffer_size	Y	Y	Y		both	yes
prepared_stmt_count			Y	Y	both	no
print-defaults	Y					
profiling			Y		session	no
profiling_history_size			Y		session	yes
protocol_version			Y		both	no
Qcache_free_blocks				Y	global	no
Qcache_free_memory				Y	global	no
Qcache_hits				Y	global	no
Qcache_inserts				Y	global	no
Qcache_lowmem_prunes				Y	global	no
Qcache_not_cached				Y	global	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
Qcache_queries_in_cache				Y	global	no
Qcache_total_blocks				Y	global	no
query_alloc_block_size	Y	Y	Y		both	yes
query_cache_limit	Y	Y	Y		both	yes
query_cache_min_res_unit	Y	Y	Y		both	yes
query_cache_size	Y	Y	Y		both	yes
query_cache_type	Y	Y	Y		both	yes
query_cache_wlock_invalidate	Y	Y	Y		both	yes
query_prealloc_size	Y	Y	Y		both	yes
Questions				Y	global	no
range_alloc_block_size	Y	Y	Y		both	yes
read_buffer_size	Y	Y	Y		both	no
read_only	Y	Y	Y		both	yes
read_rnd_buffer_size	Y	Y	Y		both	yes
record_buffer	Y	Y	Y			no
relay-log	Y	Y	Y		both	no
relay-log-index	Y	Y				
relay-log-info-file	Y	Y				
Relay_Log_File	Y	Y	Y		both	no
relay_log_purge	Y	Y	Y		both	no
Relay_Log_Space				Y	both	no
relay_log_space_limit	Y	Y	Y		both	no
Relay_Master_Log_File	Y	Y	Y		both	no
replicate-do-db	Y	Y				
replicate-do-table	Y	Y				
replicate-ignore-db	Y	Y				
replicate-ignore-table	Y	Y				
replicate-rewrite-db	Y	Y				
replicate-same-server-id	Y	Y				
replicate-wild-do-table	Y	Y				
replicate-wild-ignore-table	Y	Y				
report-host	Y	Y				
report-password	Y	Y	Y		both	no
report-port	Y	Y				
report-user	Y	Y				
rpl_recovery_rank			Y		both	yes
Rpl_status				Y	global	no
safe-mode	Y	Y				
safe-show-database	Y	Y			global	yes
- Variable: safe_show_database			Y		global	yes
safe-user-create	Y	Y				
safemalloc-mem-limit	Y	Y				
secure-auth	Y	Y			both	yes
- Variable: secure_auth			Y		both	yes

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
secure-file-priv	Y	Y			global	no
- Variable: secure_file_priv			Y		global	no
Select_full_join				Y	both	no
Select_full_range_join				Y	both	no
Select_range				Y	both	no
Select_range_check				Y	both	no
Select_scan				Y	both	no
server-id	Y	Y			both	yes
- Variable: server_id			Y		both	yes
set-variable	Y	Y				
shared-memory-base-name	Y	Y				
shared_memory	Y	Y				
show-slave-auth-info	Y	Y				
skip-automatic_sp_privileges	Y	Y				
skip-bdb	Y	Y				
- Variable: skip_bdb						
skip-character-set-client-handshake	Y	Y				
skip-concurrent-insert	Y	Y				
- Variable: skip_concurrent_insert						
skip-external-locking	Y	Y			global	no
- Variable: skip_external_locking			Y		global	no
skip-grant-tables	Y	Y				
skip-host-cache	Y	Y				
skip-innodb	Y	Y				
skip-innodb-checksums	Y	Y				
skip-isam	Y	Y	Y			no
skip-locking	Y	Y				
skip-log-warnings	Y		Y		both	no
skip-merge	Y	Y	Y		both	no
skip-name-resolve	Y	Y				
skip-networking	Y	Y			both	no
- Variable: skip_networking			Y		both	no
skip-new	Y	Y	Y		both	no
skip-plugin-innodb	Y	Y				
skip-plugin-innodb-checksums	Y	Y				
skip-safemalloc	Y	Y				
skip-show-database	Y	Y			both	no
- Variable: skip_show_database			Y		both	no
skip-slave-start	Y	Y				
skip-ssl	Y	Y				
skip-stack-trace	Y	Y				
skip-symbolic-links	Y					
skip-symlink	Y	Y				
skip-sync-bdb-logs	Y	Y	Y		both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
skip-thread-priority	Y	Y				
slave-allow-batching	Y		Y		global	yes
slave-load-tmpdir	Y	Y			both	no
- Variable: slave_load_tmpdir			Y		both	no
slave-net-timeout	Y	Y			global	yes
- Variable: slave_net_timeout			Y		global	yes
slave-skip-errors	Y	Y			both	no
- Variable: slave_skip_errors			Y		both	no
slave_compressed_protocol	Y	Y	Y		both	no
Slave_open_temp_tables				Y	global	no
Slave_retried_transactions				Y	global	no
Slave_running				Y	global	no
slave_transaction_retries	Y	Y	Y		both	yes
slow-query-log	Y	Y				
Slow_launch_threads				Y	global	no
slow_launch_time	Y	Y	Y		both	yes
Slow_queries				Y	both	no
slow_query_log_file	Y	Y	Y		global	no
socket	Y	Y	Y		both	no
sort_buffer_size	Y	Y	Y		both	yes
Sort_merge_passes				Y	both	no
Sort_range				Y	both	no
Sort_rows				Y	both	no
Sort_scan				Y	both	no
sporadic-binlog-dump-fail	Y	Y				
sql-bin-update-same	Y	Y				
sql-mode	Y	Y			both	yes
- Variable: sql_mode			Y		both	yes
sql_auto_is_null	Y	Y	Y		session	no
sql_big_selects	Y	Y	Y		both	yes
sql_big_tables			Y		session	yes
sql_buffer_result	Y	Y	Y		session	yes
sql_log_bin	Y	Y	Y		session	yes
sql_log_off	Y	Y	Y		session	yes
sql_log_update	Y	Y	Y		session	yes
sql_low_priority_updates			Y		both	yes
sql_max_join_size			Y		both	yes
sql_notes	Y	Y	Y		both	yes
sql_quote_show_create	Y	Y	Y		session	yes
sql_safe_updates	Y	Y	Y		session	yes
sql_select_limit	Y	Y	Y		session	yes
sql_slave_skip_counter			Y		global	yes
sql_warnings	Y	Y	Y		both	yes
ssl	Y	Y	Y		both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
ssl-ca	Y	Y			both	no
- Variable: ssl_ca			Y		both	no
ssl-capath	Y	Y			both	no
- Variable: ssl_capath			Y		both	no
ssl-cert	Y	Y			both	no
- Variable: ssl_cert			Y		both	no
ssl-cipher	Y	Y			both	no
- Variable: ssl_cipher			Y		both	no
ssl-key	Y	Y			both	no
- Variable: ssl_key			Y		both	no
ssl-verify-server-cert	Y	Y				
standalone	Y	Y				
storage_engine			Y		both	no
symbolic-links	Y	Y				
sync-bdb-logs	Y	Y	Y		both	no
sync-binlog	Y	Y			both	yes
- Variable: sync_binlog			Y		both	yes
sync_frm	Y	Y			both	yes
- Variable: sync_frm			Y		both	yes
sysdate-is-now	Y	Y				
system_time_zone			Y		both	no
table_cache	Y	Y	Y		both	yes
table_definition_cache			Y		global	no
table_lock_wait_timeout	Y	Y	Y		both	yes
Table_locks_immediate				Y	global	no
Table_locks_waited				Y	global	no
table_open_cache		Y	Y		global	yes
table_type			Y		both	yes
tc-heuristic-recover	Y	Y				
Tc_log_max_pages_used				Y	global	no
Tc_log_page_size				Y	global	no
Tc_log_page_waits				Y	global	no
temp-pool	Y	Y				
thread_cache_size	Y	Y	Y		both	yes
thread_concurrency	Y	Y	Y		both	no
thread_handling	Y	Y	Y		global	no
thread_stack	Y	Y	Y		both	no
Threads_cached				Y	global	no
Threads_connected				Y	global	no
Threads_created				Y	global	no
Threads_running				Y	global	no
time_format	Y	Y	Y		both	no
time_zone	Y		Y		both	yes
timed_mutexes	Y	Y	Y		both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
timestamp	Y	Y	Y		session	yes
tmp_table_size	Y	Y	Y		both	yes
tmpdir	Y	Y	Y		both	no
transaction-isolation	Y	Y				
transaction_alloc_block_size	Y	Y	Y		both	yes
transaction_prealloc_size	Y	Y	Y		both	yes
tx_isolation			Y		both	yes
unique_checks	Y	Y	Y		session	yes
updatable_views_with_limit	Y	Y	Y		both	no
Uptime				Y	global	no
Uptime_since_flush_status				Y	both	no
use-symbolic-links	Y	Y				
user	Y	Y				
verbose	Y					
version	Y		Y		both	no
version_comment			Y		both	no
version_compile_machine			Y		both	no
version_compile_os			Y		both	no
wait_timeout	Y	Y	Y		both	yes
warning_count			Y		session	no
warnings	Y	Y				

5.2.2. Command Options

When you start the `mysqld` server, you can specify program options using any of the methods described in [Section 4.3, “Specifying Program Options”](#). The most common methods are to provide options in an option file or on the command line. However, in most cases it is desirable to make sure that the server uses the same options each time it runs. The best way to ensure this is to list them in an option file. See [Section 4.3.2, “Using Option Files”](#).

MySQL Enterprise

For expert advice on setting command options, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

`mysqld` reads options from the `[mysqld]` and `[server]` groups. `mysqld_safe` reads options from the `[mysqld]`, `[server]`, `[mysqld_safe]`, and `[safe_mysqld]` groups. `mysql.server` reads options from the `[mysqld]` and `[mysql.server]` groups.

An embedded MySQL server usually reads options from the `[server]`, `[embedded]`, and `[xxxxx_SERVER]` groups, where `xxxxx` is the name of the application into which the server is embedded.

`mysqld` accepts many command options. For a brief summary, execute `mysqld --help`. To see the full list, use `mysqld -verbose --help`.

The following list shows some of the most common server options. Additional options are described in other sections:

- Options that affect security: See [Section 5.6.3, “Security-Related `mysqld` Options”](#).
- SSL-related options: See [Section 5.8.7.3, “SSL Command Options”](#).
- Binary log control options: See [Section 5.11.4, “The Binary Log”](#).
- Replication-related options: See [Section 15.1.3, “Replication Options and Variables”](#).

- Options specific to particular storage engines: See [Section 13.4.1, “MyISAM Startup Options”](#), [Section 13.5.4, “InnoDB Startup Options and System Variables”](#), and [Section 16.6.5.1, “MySQL Cluster-Related Command Options for `mysqld`”](#).

You can also set the values of server system variables by using variable names as options, as described later in this section.

- `--help, -?`

Display a short help message and exit. Use both the `--verbose` and `--help` options to see the full message.

- `--abort-slave-event-count`

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--allow-suspicious-udfs`

This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. See [Section 28.3.4.6, “User-Defined Function Security Precautions”](#).

- `--ansi`

Use standard (ANSI) SQL syntax instead of MySQL syntax. For more precise control over the server SQL mode, use the `--sql-mode` option instead. See [Section 1.9.3, “Running MySQL in ANSI Mode”](#), and [Section 5.2.6, “SQL Modes”](#).

- `--basedir=path, -b path`

The path to the MySQL installation directory. All paths are usually resolved relative to this directory.

- `big-tables`

Allow large result sets by saving all temporary sets in files. This option prevents most “table full” errors, but also slows down queries for which in-memory tables would suffice. Since MySQL 3.23.2, the server is able to handle large result sets automatically by using memory for small temporary tables and switching to disk tables where necessary.

- `--bind-address=IP`

The IP address to bind to.

- `--binlog-format={row|statement|mixed}`

Specify whether to use row-based, statement-based, or mixed replication (statement-based is default). See [Section 15.1.2, “Replication Formats”](#). This option was added in MySQL 5.1.5.

- `--binlog-row-event-max-size=N`

Specify the maximum size of a row-based binary log event, in bytes. Rows are grouped into events smaller than this size if possible. The value should be a multiple of 256. The default is 1024. See [Section 15.1.2, “Replication Formats”](#). This option was added in MySQL 5.1.5.

- `--bootstrap`

This option is used by the `mysql_install_db` script to create the MySQL privilege tables without having to start a full MySQL server.

This option is unavailable if MySQL was configured with the `--disable-grant-options` option. See [Section 2.9.2, “Typical configure Options”](#).

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--character-set-client-handshake`

Don't ignore character set information sent by the client. To ignore client information and use the default server character set, use `--`

`--skip-character-set-client-handshake`; this makes MySQL behave like MySQL 4.0.

- `--character-set-filesystem=charset_name`

The filesystem character set. This option sets the `character_set_filesystem` system variable. It was added in MySQL 5.1.6.

- `--character-set-server=charset_name, -C charset_name`

Use `charset_name` as the default server character set. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#). If you use this option to specify a non-default character set, you should also use `--collation-server` to specify the collation.

- `--chroot=path`

Put the `mysqld` server in a closed environment during startup by using the `chroot()` system call. This is a recommended security measure. Note that use of this option somewhat limits `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`.

- `--collation-server=collation_name`

Use `collation_name` as the default server collation. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--console`

(Windows only.) Write error log messages to `stderr` and `stdout` even if `--log-error` is specified. `mysqld` does not close the console window if this option is used.

- `--core-file`

Write a core file if `mysqld` dies. For some systems, you must also specify the `--core-file-size` option to `mysqld_safe`. See [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#). Note that on some systems, such as Solaris, you do not get a core file if you are also using the `--user` option.

- `--datadir=path, -h path`

The path to the data directory.

- `--debug[=debug_options], -# [debug_options]`

If MySQL is configured with `--with-debug`, you can use this option to get a trace file of what `mysqld` is doing. The `debug_options` string often is `'d:t:o,file_name'`. The default is `'d:t:i:o,mysqld.trace'`. See [MySQL Internals: Porting](#).

As of MySQL 5.1.12, using `--with-debug` to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

- `--default-character-set=charset_name (DEPRECATED)`

Use `charset_name` as the default character set. This option is deprecated in favor of `--character-set-server`. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--default-collation=collation_name`

Use `collation_name` as the default collation. This option is deprecated in favor of `--collation-server`. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--default-storage-engine=type`

Set the default storage engine (table type) for tables. See [Chapter 13, Storage Engines](#).

- `--default-table-type=type`

This option is a synonym for `--default-storage-engine`.

- `--default-time-zone=timezone`

Set the default server time zone. This option sets the global `time_zone` system variable. If this option is not given, the default time zone is the same as the system time zone (given by the value of the `system_time_zone` system variable).

- `--delay-key-write[={OFF|ON|ALL}]`

Specify how to use delayed key writes. Delayed key writing causes key buffers not to be flushed between writes for `MyISAM` tables. `OFF` disables delayed key writes. `ON` enables delayed key writes for those tables that were created with the `DELAY_KEY_WRITE` option. `ALL` delays key writes for all `MyISAM` tables. See [Section 6.5.2, “Tuning Server Parameters”](#), and [Section 13.4.1, “MyISAM Startup Options”](#).

Note

If you set this variable to `ALL`, you should not use `MyISAM` tables from within another program (such as another MySQL server or `myisamchk`) when the tables are in use. Doing so leads to index corruption.

- `--des-key-file=file_name`

Read the default DES keys from this file. These keys are used by the `DES_ENCRYPT()` and `DES_DECRYPT()` functions.

- `--disconnect-slave-event-count`

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--enable-named-pipe`

Enable support for named pipes. This option applies only on Windows NT, 2000, XP, and 2003 systems, and can be used only with the `mysqld-nt` and `mysqld-debug` servers that support named-pipe connections.

- `--event-scheduler`

Name	<code>event-scheduler</code>	
Description	Enable/disable and start/stop the event scheduler. Note that this variable underwent significant changes in behavior and permitted values in MySQL 5.1.11 and 5.1.12	
Version Introduced	5.1.6	
Option Sets Variable	Yes, <code>event_scheduler</code>	
Variable Name	<code>event-scheduler</code>	
Variable Scope	Server	
Dynamic Variable	no	
Value Set	Type	enumeration
	Default	<code>OFF</code>
	Valid Values	<code>ON, OFF, DISABLED</code>

Enable or disable, and start or stop, the event scheduler. This option was added in MySQL 5.1.6. Note that its permitted values and behaviour changed in MySQL 5.1.11, and again in MySQL 5.1.12.

For detailed information, see [The event-scheduler Option](#).

- `--exit-info[=flags], -T [flags]`

This is a bit mask of different flags that you can use for debugging the `mysqld` server. Do not use this option unless you know exactly what it does!

- `--external-locking`

Enable external locking (system locking), which is disabled by default as of MySQL 4.0. Note that if you use this option on a system on which `lockd` does not fully work (such as Linux), it is easy for `mysqld` to deadlock. This option previously was named `-enable-locking`.

For more information about external locking, including conditions under which it can and cannot be used, see [Section 6.3.4](#),

“External Locking”.

- `--flush`

Flush (synchronize) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See [Section B.1.4.2, “What to Do If MySQL Keeps Crashing”](#).

- `--general-log[={0|1}]`

Specify the initial general log state, if the `--log` or `-l` option is given. With no argument or an argument of 0, the `--general-log` option disables the log. If omitted or given with an argument of 1, the option enables the log. If `--log` or `-l` is not specified, `--general-log` has no effect. This option was added in MySQL 5.1.12.

- `--enable-pstack`

Print a symbolic stack trace on failure.

- `--gdb`

Install an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling. See [MySQL Internals: Porting](#).

- `--init-file=file_name`

Read SQL statements from this file at startup. Each statement must be on a single line and should not include comments.

This option is unavailable if MySQL was configured with the `--disable-grant-options` option. See [Section 2.9.2, “Typical configure Options”](#).

- `--innodb-xxx`

The InnoDB options are listed in [Section 13.5.4, “InnoDB Startup Options and System Variables”](#).

- `--language=lang_name, -L lang_name`

Return client error messages in the given language. `lang_name` can be given as the language name or as the full pathname to the directory where the language files are installed. See [Section 5.10.2, “Setting the Error Message Language”](#).

- `--large-pages`

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and OS. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

Currently, MySQL supports only the Linux implementation of large pages support (which is called HugeTLB in Linux). We have plans to extend this support to FreeBSD, Solaris and possibly other platforms.

Before large pages can be used on Linux, it is necessary to configure the HugeTLB memory pool. For reference, consult the `hugetlbpage.txt` file in the Linux kernel source.

This option is disabled by default.

- `--log[=file_name], -l [file_name]`

This option enables logging to the general query log, which contains entries that record client connections and SQL statements received from clients. The log output destination can be selected with the `--log-output` option as of MySQL 5.1.6. Before 5.1.6, logging occurs to the general query log file. If you omit the filename, MySQL uses `host_name.log` as the filename. See [Section 5.11.1, “Selecting General Query and Slow Query Log Output Destinations”](#), and [Section 5.11.3, “The General Query Log”](#).

- `--log-bin[=base_name]`

Enable binary logging. The server logs all statements that change data to the binary log, which is used for backup and replication. See [Section 5.11.4, “The Binary Log”](#).

The option value, if given, is the basename for the log sequence. The server creates binary log files in sequence by adding a numeric

suffix to the basename. It is recommended that you specify a basename (see [Section B.1.8.1, “Open Issues in MySQL”](#), for the reason). Otherwise, MySQL uses `host_name-bin` as the basename.

- `--log-bin-index[=file_name]`

The index file for binary log filenames. See [Section 5.11.4, “The Binary Log”](#). If you omit the filename, and if you didn't specify one with `--log-bin`, MySQL uses `host_name-bin.index` as the filename.

- `--log-bin-trust-function-creators[={0|1}]`

With no argument or an argument of 1, this option sets the `log_bin_trust_function_creators` system variable to 1. With an argument of 0, this option sets the system variable to 0. `log_bin_trust_function_creators` affects how MySQL enforces restrictions on stored function creation. See [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

- `--log-error[=file_name]`

Log errors and startup messages to this file. See [Section 5.11.2, “The Error Log”](#). If you omit the filename, MySQL uses `host_name.err`. If the filename has no extension, the server adds an extension of `.err`.

- `--log-isam[=file_name]`

Log all **MyISAM** changes to this file (used only when debugging **MyISAM**).

- `--log-long-format` (*DEPRECATED*)

Log extra information to the binary log and slow query log, if they have been activated. For example, the username and timestamp are logged for all queries. This option is deprecated, as it now represents the default logging behavior. (See the description for `--log-short-format`.) The `--log-queries-not-using-indexes` option is available for the purpose of logging queries that do not use indexes to the slow query log.

- `--log-output[=value, ...]`

This option determines the destination for general query log and slow query log output. The option value can be given as one or more of the words **TABLE**, **FILE**, or **NONE**. If the option is given without a value, the default is **TABLE** (log to the `general_log` and `slow_log` tables in the `mysql` database). **FILE** causes logging to log files. (For **FILE** logging, the `--log` and `--slow-log` options determine the log file location.) **NONE** disables logging. If **NONE** is present in the option value, it takes precedence over any other words that are present. **TABLE** and **FILE** can both be given to select to both log output destinations.

This option selects log output destinations, but does not enable log output. To do that, use the `--log` and `--log-slow-queries` options. For more information, see [Section 5.11.1, “Selecting General Query and Slow Query Log Output Destinations”](#).

The `--log-output` option was added in MySQL 5.1.6.

- `--log-queries-not-using-indexes`

If you are using this option with `--log-slow-queries`, queries that do not use indexes are logged to the slow query log. See [Section 5.11.5, “The Slow Query Log”](#).

- `--log-short-format`

Log less information to the binary log and slow query log, if they have been activated. For example, the username and timestamp are not logged for queries.

- `--log-slow-admin-statements`

Log slow administrative statements such as **OPTIMIZE TABLE**, **ANALYZE TABLE**, and **ALTER TABLE** to the slow query log.

- `--log-slow-queries[=file_name]`

This option enables logging to the slow query log, which contains entries for all queries that have taken more than `long_query_time` seconds to execute. See the descriptions of the `--log-long-format` and `--log-short-format` options for details.

The log output destination can be selected with the `--log-output` option as of MySQL 5.1.6. Before 5.1.6, logging occurs to the

slow query log file. If you omit the filename, MySQL uses `host_name-slow.log` as the filename. See [Section 5.11.1](#), “Selecting General Query and Slow Query Log Output Destinations”, and [Section 5.11.5](#), “The Slow Query Log”.

- `--log-tc=file_name`

The name of the memory-mapped transaction coordinator log file (for XA transactions that affect multiple storage engines when the binary log is disabled). The default name is `tc.log`. The file is created under the data directory if not given as a full pathname. Currently, this option is unused.

- `--log-tc-size=size`

The size in bytes of the memory-mapped transaction coordinator log. The default size is 24KB.

- `--log-warnings[=level], -w [level]`

Print out warnings such as `Aborted connection...` to the error log. Enabling this option is recommended, for example, if you use replication (you get more information about what is happening, such as messages about network failures and reconnections). This option is enabled (1) by default, and the default `level` value if omitted is 1. To disable this option, use `--log-warnings=0`. Aborted connections are not logged to the error log unless the value is greater than 1. See [Section B.1.2.10](#), “Communication Errors and Aborted Connections”.

- `--low-priority-updates`

Give table-modifying operations (`INSERT`, `REPLACE`, `DELETE`, `UPDATE`) lower priority than selects. This can also be done via `{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...` to lower the priority of only one query, or by `SET LOW_PRIORITY_UPDATES=1` to change the priority in one thread. This affects only storage engines that use only table-level locking (`MyISAM`, `MEMORY`, `MERGE`). See [Section 6.3.2](#), “Table Locking Issues”.

- `--max-binlog-dump-events`

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--mlock`

Lock the `mysqld` process in memory. This works on systems such as Solaris that support the `mlockall()` system call. This might help if you have a problem where the operating system is causing `mysqld` to swap on disk. Note that use of this option requires that you run the server as `root`, which is normally not a good idea for security reasons. See [Section 5.6.5](#), “How to Run MySQL as a Normal User”.

- `--myisam-recover[=option[,option]...]`

Set the `MyISAM` storage engine recovery mode. The option value is any combination of the values of `DEFAULT`, `BACKUP`, `FORCE`, or `QUICK`. If you specify multiple values, separate them by commas. You can also use a value of `"` to disable this option. If this option is used, each time `mysqld` opens a `MyISAM` table, it checks whether the table is marked as crashed or wasn't closed properly. (The last option works only if you are running with external locking disabled.) If this is the case, `mysqld` runs a check on the table. If the table was corrupted, `mysqld` attempts to repair it.

The following options affect how the repair works:

Option	Description
<code>DEFAULT</code>	The same as not giving any option to <code>--myisam-recover</code> .
<code>BACKUP</code>	If the data file was changed during recovery, save a backup of the <code>tbl_name.MYD</code> file as <code>tbl_name-datetime.BAK</code> .
<code>FORCE</code>	Run recovery even if we would lose more than one row from the <code>.MYD</code> file.
<code>QUICK</code>	Don't check the rows in the table if there aren't any delete blocks.

Before the server automatically repairs a table, it writes a note about the repair to the error log. If you want to be able to recover from most problems without user intervention, you should use the options `BACKUP`, `FORCE`. This forces a repair of a table even if some rows would be deleted, but it keeps the old data file as a backup so that you can later examine what happened.

See [Section 13.4.1](#), “MyISAM Startup Options”.

- `--ndb-cluster-connection-pool=value`

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

This option controls how many connections are made to a MySQL Cluster by a MySQL server acting as a cluster SQL node. Normally, `mysqld` uses only a single connection to the cluster, and the usual way to scale out is by adding more SQL nodes (`mysqld` instances). However, by setting this option to a value greater than 1 (the default), it is possible to scale out by allowing multiple MySQL clients to leverage multiple connections made to the cluster by a single MySQL server. The maximum possible is 63.

Important

When this option is set to a value *N*, there must be at least *N* `[mysqld]` or `[api]` slots available for use; if there are fewer than this number, no MySQL clients are able to connect to the MySQL server.

This option was added in MySQL 5.1.19/NDB-6.2.2.

The following information applies to all MySQL Cluster users.

- `--ndb-connectstring=connect_string`

When using the `NDB` storage engine, it is possible to point out the management server that distributes the cluster configuration by setting the connect string option. See Section 16.4.4.2, “The Cluster Connectstring”, for syntax.

- `--ndbcluster`

If the binary includes support for the `NDB Cluster` storage engine, this option enables the engine, which is disabled by default. See Chapter 16, *MySQL Cluster*.

- `--old-passwords`

Force the server to generate short (pre-4.1) password hashes for new passwords. This is useful for compatibility when the server must support older client programs. See Section 5.7.9, “Password Hashing as of MySQL 4.1”.

- `--one-thread`

Only use one thread (for debugging under Linux). This option is available only if the server is built with debugging enabled. See *MySQL Internals: Porting*.

As of MySQL 5.1.17, this option is deprecated; use `--thread_handling=one-thread` instead.

- `--open-files-limit=count`

Change the number of file descriptors available to `mysqld`. If this option is not set or is set to 0, `mysqld` uses the value to reserve file descriptors with `setrlimit()`. If the value is 0, `mysqld` reserves `max_connections×5` or `max_connections + table_open_cache×2` files (whichever is larger). You should try increasing this value if `mysqld` gives you the error `Too many open files`.

- `--pid-file=path`

The pathname of the process ID file. This file is used by other programs such as `mysqld_safe` to determine the server's process ID.

- `--port=port_num, -P port_num`

The port number to use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the `root` system user.

- `--port-open-timeout=num`

On some systems, when the server is stopped, the TCP/IP port might not become available immediately. If the server is restarted quickly afterward, its attempt to reopen the port can fail. This option indicates how many seconds the server should wait for the TCP/IP port to become free if it cannot be opened. The default is not to wait. This option was added in MySQL 5.1.5.

- `--safe-mode`

Skip some optimization stages.

- `--safe-show-database` (*DEPRECATED*)

See [Section 5.7.3, “Privileges Provided by MySQL”](#).

- `--safe-user-create`

If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement unless the user has the `INSERT` privilege for the `mysql.user` table or any column in the table. If you want a user to have the ability to create new users that have those privileges that the user has the right to grant, you should grant the user the following privilege:

```
GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

This ensures that the user cannot change any privilege columns directly, but has to use the `GRANT` statement to give privileges to other users.

- `--secure-auth`

Disallow authentication by clients that attempt to use accounts that have old (pre-4.1) passwords.

- `--secure-file-priv=path`

This option limits the effect of the `LOAD_FILE()` function and the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements to work only with files in the specified directory.

This option was added in MySQL 5.1.17.

- `--shared-memory`

Enable shared-memory connections by local clients. This option is available only on Windows.

- `--shared-memory-base-name=name`

The name of shared memory to use for shared-memory connections. This option is available only on Windows. The default name is `MYSQL`. The name is case sensitive.

- `--skip-concurrent-insert`

Turn off the ability to select and insert at the same time on `MyISAM` tables. (This is to be used only if you think you have found a bug in this feature.) See [Section 6.3.3, “Concurrent Inserts”](#).

- `--skip-external-locking`

Do not use external locking (system locking). For more information about external locking, including conditions under which it can and cannot be used, see [Section 6.3.4, “External Locking”](#).

External locking has been disabled by default since MySQL 4.0.

- `--skip-grant-tables`

This option causes the server not to use the privilege system at all, which gives anyone with access to the server *unrestricted access to all databases*. You can cause a running server to start using the grant tables again by executing `mysqladmin flush-privileges` or `mysqladmin reload` command from a system shell, or by issuing a MySQL `FLUSH PRIVILEGES` statement after connecting to the server. This option also suppresses loading of plugins and user-defined functions (UDFs). Beginning with MySQL 5.1.17, it also suppresses loading of scheduled events ([Bug#28607](#)).

This option is unavailable if MySQL was configured with the `--disable-grant-options` option. See [Section 2.9.2, “Typical configure Options”](#).

- `--skip-host-cache`

Do not use the internal hostname cache for faster name-to-IP resolution. Instead, query the DNS server every time a client connects. See [Section 6.5.9, “How MySQL Uses DNS”](#).

- `--skip-innodb`

Disable the **InnoDB** storage engine. This saves memory and disk space and might speed up some operations. Do not use this option if you require **InnoDB** tables.

- `--skip-name-resolve`

Do not resolve hostnames when checking client connections. Use only IP numbers. If you use this option, all **Host** column values in the grant tables must be IP numbers or **localhost**. See [Section 6.5.9, “How MySQL Uses DNS”](#).

- `--skip-ndbcluster`

Disable the **NDB Cluster** storage engine. This is the default for binaries that were built with **NDB Cluster** storage engine support; the server allocates memory and other resources for this storage engine only if the `--ndbcluster` option is given explicitly. See [Section 16.4.3, “Quick Test Setup of MySQL Cluster”](#), for an example of usage.

- `--skip-networking`

Don't listen for TCP/IP connections at all. All interaction with **mysqld** must be made via named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are allowed. See [Section 6.5.9, “How MySQL Uses DNS”](#).

- `--sporadic-binlog-dump-fail`

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--ssl*`

Options that begin with `--ssl` specify whether to allow clients to connect via SSL and indicate where to find SSL keys and certificates. See [Section 5.8.7.3, “SSL Command Options”](#).

- `--standalone`

Available on Windows NT-based systems only; instructs the MySQL server not to run as a service.

- `--symbolic-links`, `--skip-symbolic-links`

Enable or disable symbolic link support. This option has different effects on Windows and Unix:

- On Windows, enabling symbolic links allows you to establish a symbolic link to a database directory by creating a `db_name.sym` file that contains the path to the real directory. See [Section 6.6.1.3, “Using Symbolic Links for Databases on Windows”](#).
- On Unix, enabling symbolic links means that you can link a **MyISAM** index file or data file to another directory with the **INDEX DIRECTORY** or **DATA DIRECTORY** options of the **CREATE TABLE** statement. If you delete or rename the table, the files that its symbolic links point to also are deleted or renamed. See [Section 6.6.1.2, “Using Symbolic Links for Tables on Unix”](#).

- `--skip-safemalloc`

If MySQL is configured with `--with-debug=full`, all MySQL programs check for memory overruns during each memory allocation and memory freeing operation. This checking is very slow, so for the server you can avoid it when you don't need it by using the `--skip-safemalloc` option.

- `--skip-show-database`

With this option, the **SHOW DATABASES** statement is allowed only to users who have the **SHOW DATABASES** privilege, and the statement displays all database names. Without this option, **SHOW DATABASES** is allowed to all users, but displays each database name only if the user has the **SHOW DATABASES** privilege or some privilege for the database. Note that *any* global privilege is considered a privilege for the database.

- `--skip-stack-trace`

Don't write stack traces. This option is useful when you are running **mysqld** under a debugger. On some systems, you also must use this option to get a core file. See [MySQL Internals: Porting](#).

- `--skip-thread-priority`

Disable using thread priorities for faster response time.

- `--slow-query-log[={0|1}]`

Specify the initial slow query log state, if the `--log-slow-queries` option is given. With no argument or an argument of 0, the `--slow-query-log` option disables the log. If omitted or given with an argument of 1, the option enables the log. If `--log` or `--log-slave-statements` is not specified, `--slow-query-log` has no effect. This option was added in MySQL 5.1.12.

- `--socket=path`

Name	<code>socket</code>	
Description	Socket file to use for connection	
Option Sets Variable	Yes, <code>socket</code>	
Variable Name	<code>socket</code>	
Variable Scope	Server	
Dynamic Variable	no	
Value Set	Type	linux
	Default	<code>/tmp/mysql.sock</code>
Value Set	Type	hpux
	Default	<code>/tmp/mysql.sock</code>
Value Set	Type	solaris
	Default	<code>/tmp/mysql.sock</code>
Value Set	Type	macosx
	Default	<code>/tmp/mysql.sock</code>

On Unix, this option specifies the Unix socket file to use when listening for local connections. The default value is `/tmp/mysql.sock`. On Windows, the option specifies the pipe name to use when listening for local connections that use a named pipe. The default value is `MySQL` (not case sensitive).

- `--sql-mode=value[,value[,value...]]`

Set the SQL mode. See [Section 5.2.6, “SQL Modes”](#).

- `--sysdate-is-now`

`SYSDATE()` by default returns the time at which it executes, not the time at which the statement in which it occurs begins executing. This differs from the behavior of `NOW()`. This option causes `SYSDATE()` to be an alias for `NOW()`. For information about the implications for binary logging and replication, see the description for `SYSDATE()` in [Section 11.6, “Date and Time Functions”](#) and for `SET TIMESTAMP` in [Section 12.5.3, “SET Syntax”](#).

This option was added in MySQL 5.1.8.

- `--tc-heuristic-recover={COMMIT|ROLLBACK}`

The type of decision to use in the heuristic recovery process. Currently, this option is unused.

- `--temp-pool`

This option causes most temporary files created by the server to use a small set of names, rather than a unique name for each new file. This works around a problem in the Linux kernel dealing with creating many new files with different names. With the old behavior, Linux seems to “leak” memory, because it is being allocated to the directory entry cache rather than to the disk cache.

- `--transaction-isolation=level`

Sets the default transaction isolation level. The `level` value can be `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, or `SERIALIZABLE`. See [Section 12.4.6, “SET TRANSACTION Syntax”](#).

- `--tmpdir=path, -t path`

The path of the directory to use for creating temporary files. It might be useful if your default `/tmp` directory resides on a partition that is too small to hold temporary tables. This option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (':') on Unix and semicolon characters(';') on Windows, NetWare, and OS/2. If the MySQL server is acting as a replication slave, you should not set `--tmpdir` to point to a directory on a memory-based filesystem or to a directory that is cleared when the server host restarts. For more information about the storage location of temporary files, see [Section B.1.4.4, “Where MySQL Stores Temporary Files”](#). A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails.

- `--user={user_name|user_id}, -u {user_name|user_id}`

Run the `mysqld` server as the user having the name `user_name` or the numeric user ID `user_id`. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

This option is *mandatory* when starting `mysqld` as `root`. The server changes its user ID during its startup sequence, causing it to run as that particular user rather than as `root`. See [Section 5.6.1, “General Security Guidelines”](#).

To avoid a possible security hole where a user adds a `--user=root` option to a `my.cnf` file (thus causing the server to run as `root`), `mysqld` uses only the first `--user` option specified and produces a warning if there are multiple `--user` options. Options in `/etc/my.cnf` and `$MYSQL_HOME/my.cnf` are processed before command-line options, so it is recommended that you put a `--user` option in `/etc/my.cnf` and specify a value other than `root`. The option in `/etc/my.cnf` is found before any other `--user` options, which ensures that the server runs as a user other than `root`, and that a warning results if any other `--user` option is found.

- `--version, -V`

Display version information and exit.

You can assign a value to a server system variable by using an option of the form `--var_name=value`. For example, `--key_buffer_size=32M` sets the `key_buffer_size` variable to a value of 32MB.

Note that when you assign a value to a variable, MySQL might automatically correct the value to stay within a given range, or adjust the value to the closest allowable value if only certain values are allowed.

If you want to restrict the maximum value to which a variable can be set at runtime with `SET`, you can define this by using the `--maximum-var_name=value` command-line option.

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated.*

You can change the values of most system variables for a running server with the `SET` statement. See [Section 12.5.3, “SET Syntax”](#).

[Section 5.2.3, “System Variables”](#), provides a full description for all variables, and additional information for setting them at server startup and runtime. [Section 6.5.2, “Tuning Server Parameters”](#), includes information on optimizing the server by tuning system variables.

5.2.3. System Variables

The MySQL server maintains many system variables that indicate how it is configured. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

There are several ways to see the names and values of system variables:

- To see the values that a server will use based on its compiled-in defaults and any option files that it reads, use this command:

```
mysqld --verbose --help
```

- To see the values that a server will use based on its compiled-in defaults, ignoring the settings in any option files, use this command:

```
mysqld --no-defaults --verbose --help
```

- To see the current values used by a running server, use the [SHOW VARIABLES](#) statement.

This section provides a description of each system variable. Variables with no version indicated are present in all MySQL 5.1 releases. For historical information concerning their implementation, please see *MySQL 5.0 Reference Manual* and *MySQL 3.23, 4.0, 4.1 Reference Manual*.

The following table lists all available system variables:

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
auto-increment-increment	Y	Y			both	no
- Variable: auto_increment_increment			Y		both	no
auto-increment-offset	Y	Y			both	no
- Variable: auto_increment_offset			Y		both	no
autocommit	Y	Y	Y		session	yes
automatic_sp_privileges	Y	Y	Y		both	no
back_log	Y	Y	Y		both	no
basedir	Y	Y	Y		both	no
big-tables	Y	Y			both	yes
- Variable: big_tables			Y		both	yes
binlog_cache_size	Y	Y	Y		both	yes
binlog_format	Y	Y	Y		both	no
bulk_insert_buffer_size	Y	Y	Y		both	yes
character-set-filesystem	Y	Y			both	no
- Variable: character_set_filesystem			Y		both	no
character-set-server	Y	Y			both	yes
- Variable: character_set_server			Y		both	yes
character-sets-dir	Y	Y			both	no
- Variable: character_sets_dir			Y		both	no
character_set_client	Y	Y	Y		both	yes
character_set_connection	Y	Y	Y		both	yes
character_set_database	Y	Y	Y		both	yes
character_set_results	Y	Y	Y		both	yes
character_set_system	Y	Y	Y		both	yes
collation_connection	Y	Y	Y		both	yes
collation_database	Y	Y	Y		both	yes
collation_server	Y	Y	Y		both	yes
completion_type	Y	Y	Y		both	yes
concurrent-insert	Y	Y			both	yes
- Variable: concurrent_insert			Y		both	yes
connect_timeout	Y	Y	Y		both	yes
datadir	Y	Y	Y		both	no
date_format	Y	Y	Y		both	yes
datetime_format	Y	Y	Y		both	yes
default_week_format	Y	Y	Y		both	yes
delay-key-write	Y	Y			both	yes
- Variable: delay_key_write			Y		both	yes
delayed_insert_limit	Y	Y	Y		both	yes

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
delayed_insert_timeout	Y	Y	Y		both	yes
delayed_queue_size	Y	Y	Y		both	yes
des-key-file	Y	Y	Y		both	no
div_precision_increment	Y	Y	Y		both	yes
engine-condition-pushdown	Y	Y			both	yes
- Variable: engine_condition_pushdown			Y		both	yes
error_count	Y	Y	Y		session	no
event-scheduler	Y	Y				no
- Variable: event_scheduler			Y			no
expire_logs_days	Y	Y	Y		global	yes
flush	Y	Y	Y		both	yes
flush_time	Y	Y	Y		both	yes
foreign_key_checks	Y	Y	Y		session	yes
ft_boolean_syntax	Y	Y	Y		both	yes
ft_max_word_len	Y	Y	Y		both	yes
ft_min_word_len	Y	Y	Y		global	yes
ft_query_expansion_limit	Y	Y	Y		both	yes
ft_stopword_file	Y	Y	Y		both	yes
general-log	Y	Y			global	no
- Variable: general_log			Y		global	no
general_log_file	Y	Y	Y		global	no
group_concat_max_len	Y	Y	Y		both	yes
have_archive	Y	Y	Y		global	no
have_blackhole_engine	Y	Y	Y		global	no
have_compress	Y	Y	Y		global	no
have_crypt	Y	Y	Y		global	no
have_csv	Y	Y	Y		global	no
have_dynamic_loading	Y	Y	Y		global	no
have_example_engine	Y	Y	Y		global	no
have_federated_engine	Y	Y	Y		global	no
have_geometry	Y	Y	Y		global	no
have_innodb	Y	Y	Y		global	no
have_isam	Y	Y	Y		global	no
have_merge_engine	Y	Y	Y		global	no
have_ndbcluster	Y	Y	Y		global	no
have_openssl	Y	Y	Y		global	no
have_partitioning	Y	Y	Y		global	no
have_query_cache	Y	Y	Y		global	no
have_raid	Y	Y	Y		global	no
have_row_based_replication	Y	Y	Y		global	no
have_rtree_keys	Y	Y	Y		global	no
have_symlink	Y	Y	Y		global	no
hostname	Y	Y	Y		global	no
identity	Y	Y	Y		session	yes

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
init-file	Y	Y			both	no
- Variable: init_file			Y		both	no
init_connect	Y	Y	Y		global	no
init_slave	Y	Y	Y		both	no
innodb_additional_mem_pool_size	Y	Y	Y		both	no
innodb_autoextend_increment	Y	Y	Y		both	yes
innodb_autoinc_lock_mode	Y	Y	Y		global	yes
innodb_buffer_pool_ave_mem_mb	Y	Y	Y		both	no
innodb_buffer_pool_size	Y	Y	Y		both	no
innodb_checksums	Y	Y	Y		both	yes
innodb_commit_concurrency	Y	Y	Y		both	yes
innodb_concurrency_tickets	Y	Y	Y		both	yes
innodb_data_file_path	Y	Y	Y		both	no
innodb_data_home_dir	Y	Y	Y		both	no
innodb_doublewrite	Y	Y	Y		both	no
innodb_fast_shutdown	Y	Y	Y		both	no
innodb_file_io_threads	Y	Y	Y		both	no
innodb_file_per_table	Y	Y	Y		both	no
innodb_flush_log_at_trx_commit	Y	Y	Y		global	yes
innodb_flush_method	Y	Y	Y		both	no
innodb_force_recovery	Y	Y	Y		both	no
innodb_lock_wait_timeout	Y	Y	Y		both	no
innodb_locks_unsafe_for_binlog	Y	Y	Y		both	no
innodb_log_arch_dir	Y	Y	Y		both	no
innodb_log_archive	Y	Y	Y		both	no
innodb_log_buffer_size	Y	Y	Y		both	no
innodb_log_file_size	Y	Y	Y		both	no
innodb_log_files_in_group	Y	Y	Y		both	no
innodb_log_group_home_dir	Y	Y	Y		both	no
innodb_max_dirty_pages_pct	Y	Y	Y		both	yes
innodb_max_purge_lag	Y	Y	Y		both	yes
innodb_mirrored_log_groups	Y	Y	Y		both	no
innodb_open_files	Y	Y	Y		both	no
innodb_rollback_on_timeout	Y	Y	Y		both	no
innodb_stats_on_metadata	Y	Y	Y			no
innodb_status_file	Y	Y	Y		both	no
innodb_support_xa	Y	Y	Y		both	yes
innodb_sync_spin_loops	Y	Y	Y		both	yes
innodb_table_locks	Y	Y	Y		both	yes
innodb_thread_concurrency	Y	Y	Y		global	yes
innodb_thread_sleep_delay	Y	Y	Y		both	yes
insert_id	Y	Y	Y		session	yes
interactive_timeout	Y	Y	Y		both	yes
join_buffer_size	Y	Y	Y		both	yes

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
keep_files_on_create	Y	Y	Y		global	yes
key_buffer_size	Y	Y	Y		both	yes
key_cache_age_threshold	Y	Y	Y		both	no
key_cache_block_size	Y	Y	Y		both	no
key_cache_division_limit	Y	Y	Y		both	no
language	Y	Y	Y		both	no
large-pages	Y	Y			both	no
- Variable: large_pages			Y		both	no
last_insert_id	Y	Y	Y		session	yes
lc_time_names	Y	Y	Y		both	yes
license	Y	Y	Y		both	no
local-infile	Y	Y			global	yes
- Variable: local_infile			Y		global	yes
log	Y	Y	Y		both	no
log-bin	Y	Y			both	no
- Variable: log_bin			Y		both	no
log-bin-index	Y	Y	Y		both	no
log-bin-trust-function-creators	Y	Y			both	yes
- Variable: log_bin_trust_function_creators			Y		both	yes
log-error	Y	Y			both	no
- Variable: log_error			Y		both	no
log-isam	Y	Y	Y		both	no
log-queries-not-using-indexes	Y	Y			both	yes
- Variable: log_queries_not_using_indexes			Y		both	yes
log-slave-updates	Y	Y			both	no
- Variable: log_slave_updates			Y		both	no
log-slow-queries	Y	Y			both	no
- Variable: log_slow_queries			Y		both	no
log-warnings	Y	Y			both	yes
- Variable: log_warnings			Y		both	yes
log_output	Y	Y	Y		global	yes
long_query_time	Y	Y	Y		both	yes
low-priority-updates	Y	Y			both	yes
- Variable: low_priority_updates			Y		both	yes
lower_case_file_system	Y	Y	Y		both	no
lower_case_table_names	Y	Y	Y		both	no
master-connect-retry	Y	Y	Y			no
Master_Log_File	Y	Y	Y		both	no
Master_SSL_Allowed	Y	Y	Y		both	no
Master_SSL_CA_File	Y	Y	Y		both	no
Master_SSL_CA_Path	Y	Y	Y		both	no
max_allowed_packet	Y	Y	Y		both	yes
max_binlog_cache_size	Y	Y	Y		global	yes
max_binlog_size	Y	Y	Y		global	yes

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
max_connect_errors	Y	Y	Y		global	yes
max_connections	Y	Y	Y		global	yes
max_delayed_threads	Y	Y	Y		global	yes
max_error_count	Y	Y	Y		both	yes
max_heap_table_size	Y	Y	Y		both	yes
max_insert_delayed_threads	Y	Y	Y		global	yes
max_join_size	Y	Y	Y		both	yes
max_length_for_sort_data	Y	Y	Y		both	yes
max_prepared_stmt_count	Y	Y	Y		global	yes
max_relay_log_size	Y	Y	Y		global	yes
max_seeks_for_key	Y	Y	Y		both	yes
max_sort_length	Y	Y	Y		both	yes
max_sp_recursion_depth	Y	Y	Y		global	yes
max_tmp_tables	Y	Y	Y		both	yes
max_user_connections	Y	Y	Y		both	yes
max_write_lock_count	Y	Y	Y		global	yes
memlock	Y	Y			both	no
- Variable: locked_in_memory			Y		both	no
merge	Y	Y	Y		both	no
multi_range_count	Y	Y	Y		both	yes
multi_read_range	Y	Y	Y		both	yes
myisam-recover	Y	Y	Y		both	no
myisam_block_size	Y	Y	Y		both	yes
myisam_data_pointer_size	Y	Y	Y		global	yes
myisam_max_sort_file_size	Y	Y	Y		both	yes
myisam_repair_threads	Y	Y	Y		both	yes
myisam_sort_buffer_size	Y	Y	Y		both	yes
myisam_stats_method	Y	Y	Y		both	yes
myisam_use_mmap	Y	Y	Y		global	no
ndb_autoincrement_prefetch_sz	Y	Y	Y		both	yes
ndb_cache_check_time	Y	Y	Y		both	yes
ndb_cluster_connection_pool	Y	Y	Y			no
ndb_extra_logging	Y	Y	Y		global	yes
ndb_force_send	Y	Y	Y		both	yes
ndb_use_copying_alter_table	Y	Y	Y			no
ndb_use_exact_count	Y	Y	Y		both	yes
ndb_wait_connected	Y	Y	Y			no
ndbcluster	Y	Y	Y		both	yes
net_buffer_length	Y	Y	Y		both	yes
net_read_timeout	Y	Y	Y		both	yes
net_retry_count	Y	Y	Y		both	yes
net_write_timeout	Y	Y	Y		both	yes
new	Y	Y	Y		both	no
old	Y	Y	Y			no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
old-passwords	Y	Y			both	yes
- Variable: old_passwords			Y		both	yes
open_files_limit	Y	Y	Y		both	no
optimizer_prune_level	Y	Y	Y		both	yes
optimizer_search_depth	Y	Y	Y		both	yes
pid-file	Y	Y	Y		both	no
plugin-innodb-doublewrite	Y	Y			both	no
- Variable: plugin_innodb_doublewrite			Y		both	no
plugin_dir	Y	Y	Y			no
plugin_innodb_additional_mem_pool_size	Y	Y	Y		both	no
plugin_innodb_autoextend_increment	Y	Y	Y		both	yes
plugin_innodb_buffer_pool_ave_mem_mb	Y	Y			both	no
- Variable: innodb_buffer_pool_ave_mem_mb			Y		both	no
plugin_innodb_buffer_pool_size	Y	Y	Y		both	no
plugin_innodb_checksums	Y	Y	Y		both	yes
plugin_innodb_commit_concurrency	Y	Y	Y		both	yes
plugin_innodb_concurrency_tickets	Y	Y	Y		both	yes
plugin_innodb_data_file_path	Y	Y	Y		both	no
plugin_innodb_data_home_dir	Y	Y	Y		both	no
plugin_innodb_fast_shutdown	Y	Y	Y		both	no
plugin_innodb_file_io_threads	Y	Y	Y		both	no
plugin_innodb_file_per_table	Y	Y	Y		both	no
plugin_innodb_flush_log_at_trx_commit	Y	Y	Y		global	yes
plugin_innodb_flush_method	Y	Y	Y		both	no
plugin_innodb_force_recovery	Y	Y	Y		both	no
plugin_innodb_lock_wait_timeout	Y	Y	Y		both	no
plugin_innodb_locks_unsafe_for_binlog	Y	Y	Y		both	no
plugin_innodb_log_archive	Y	Y	Y		both	no
plugin_innodb_log_buffer_size	Y	Y	Y		both	no
plugin_innodb_log_file_size	Y	Y	Y		both	no
plugin_innodb_log_files_in_group	Y	Y	Y		both	no
plugin_innodb_log_group_home_dir	Y	Y	Y		both	no
plugin_innodb_max_dirty_pages_pct	Y	Y	Y		both	yes
plugin_innodb_max_purge_lag	Y	Y	Y		both	yes
plugin_innodb_mirrored_log_groups	Y	Y	Y		both	no
plugin_innodb_open_files	Y	Y	Y		both	no
plugin_innodb_rollback_on_timeout	Y	Y	Y			no
plugin_innodb_stats_on_metadata	Y	Y	Y			no
plugin_innodb_status_file	Y	Y	Y			no
plugin_innodb_support_xa	Y	Y	Y		both	yes
plugin_innodb_sync_spin_loops	Y	Y			both	yes
- Variable: innodb_sync_spin_loops			Y		both	yes
plugin_innodb_table_locks	Y	Y	Y		both	yes
plugin_innodb_thread_concurrency	Y	Y	Y		both	yes

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
plugin_innodb_thread_sleep_delay	Y	Y	Y		both	yes
port	Y	Y	Y		both	no
preload_buffer_size	Y	Y	Y		both	yes
prepared_stmt_count	Y	Y	Y	Y	both	no
protocol_version	Y	Y	Y		both	no
query_alloc_block_size	Y	Y	Y		both	yes
query_cache_limit	Y	Y	Y		both	yes
query_cache_min_res_unit	Y	Y	Y		both	yes
query_cache_size	Y	Y	Y		both	yes
query_cache_type	Y	Y	Y		both	yes
query_cache_wlock_invalidate	Y	Y	Y		both	yes
query_prealloc_size	Y	Y	Y		both	yes
range_alloc_block_size	Y	Y	Y		both	yes
read_buffer_size	Y	Y	Y		both	no
read_only	Y	Y	Y		both	yes
read_rnd_buffer_size	Y	Y	Y		both	yes
relay-log	Y	Y	Y		both	no
Relay_Log_File	Y	Y	Y		both	no
relay_log_purge	Y	Y	Y		both	no
relay_log_space_limit	Y	Y	Y		both	no
Relay_Master_Log_File	Y	Y	Y		both	no
report-password	Y	Y	Y		both	no
rpl_recovery_rank	Y	Y	Y		both	yes
secure-auth	Y	Y			both	yes
- Variable: secure_auth			Y		both	yes
secure-file-priv	Y	Y			global	no
- Variable: secure_file_priv			Y		global	no
server-id	Y	Y			both	yes
- Variable: server_id			Y		both	yes
skip-external-locking	Y	Y			global	no
- Variable: skip_external_locking			Y		global	no
skip-log-warnings	Y	Y	Y		both	no
skip-merge	Y	Y	Y		both	no
skip-networking	Y	Y			both	no
- Variable: skip_networking			Y		both	no
skip-new	Y	Y	Y		both	no
skip-show-database	Y	Y			both	no
- Variable: skip_show_database			Y		both	no
slave-allow-batching	Y	Y	Y		global	yes
slave-load-tmpdir	Y	Y			both	no
- Variable: slave_load_tmpdir			Y		both	no
slave-net-timeout	Y	Y			global	yes
- Variable: slave_net_timeout			Y		global	yes
slave-skip-errors	Y	Y			both	no

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
- Variable: slave_skip_errors			Y		both	no
slave_compressed_protocol	Y	Y	Y		both	no
slave_transaction_retries	Y	Y	Y		both	yes
slow_launch_time	Y	Y	Y		both	yes
slow_query_log_file	Y	Y	Y		global	no
socket	Y	Y	Y		both	no
sort_buffer_size	Y	Y	Y		both	yes
sql-mode	Y	Y			both	yes
- Variable: sql_mode			Y		both	yes
sql_auto_is_null	Y	Y	Y		session	no
sql_big_selects	Y	Y	Y		both	yes
sql_big_tables	Y	Y	Y		session	yes
sql_buffer_result	Y	Y	Y		session	yes
sql_log_bin	Y	Y	Y		session	yes
sql_log_off	Y	Y	Y		session	yes
sql_log_update	Y	Y	Y		session	yes
sql_low_priority_updates	Y	Y	Y		both	yes
sql_max_join_size	Y	Y	Y		both	yes
sql_notes	Y	Y	Y		both	yes
sql_quote_show_create	Y	Y	Y		session	yes
sql_safe_updates	Y	Y	Y		session	yes
sql_select_limit	Y	Y	Y		session	yes
sql_slave_skip_counter	Y	Y	Y		global	yes
sql_warnings	Y	Y	Y		both	yes
ssl	Y	Y	Y		both	no
ssl-ca	Y	Y			both	no
- Variable: ssl_ca			Y		both	no
ssl-capath	Y	Y			both	no
- Variable: ssl_capath			Y		both	no
ssl-cert	Y	Y			both	no
- Variable: ssl_cert			Y		both	no
ssl-cipher	Y	Y			both	no
- Variable: ssl_cipher			Y		both	no
ssl-key	Y	Y			both	no
- Variable: ssl_key			Y		both	no
storage_engine	Y	Y	Y		both	no
sync-binlog	Y	Y			both	yes
- Variable: sync_binlog			Y		both	yes
sync_frm	Y	Y			both	yes
- Variable: sync_frm			Y		both	yes
system_time_zone	Y	Y	Y		both	no
table_cache	Y	Y	Y		both	yes
table_definition_cache	Y	Y	Y		global	no
table_lock_wait_timeout	Y	Y	Y		both	yes

Name	Cmd-line	Option file	System Var	Status Var	Var Scope	Dynamic
table_open_cache	Y	Y	Y		global	yes
table_type	Y	Y	Y		both	yes
thread_cache_size	Y	Y	Y		both	yes
thread_concurrency	Y	Y	Y		both	no
thread_handling	Y	Y	Y		global	no
thread_stack	Y	Y	Y		both	no
time_format	Y	Y	Y		both	no
time_zone	Y	Y	Y		both	yes
timed_mutexes	Y	Y	Y		both	no
timestamp	Y	Y	Y		session	yes
tmp_table_size	Y	Y	Y		both	yes
tmpdir	Y	Y	Y		both	no
transaction_alloc_block_size	Y	Y	Y		both	yes
transaction_prealloc_size	Y	Y	Y		both	yes
tx_isolation	Y	Y	Y		both	yes
unique_checks	Y	Y	Y		session	yes
updatable_views_with_limit	Y	Y	Y		both	no
version	Y	Y	Y		both	no
version_comment	Y	Y	Y		both	no
version_compile_machine	Y	Y	Y		both	no
version_compile_os	Y	Y	Y		both	no
wait_timeout	Y	Y	Y		both	yes
warning_count	Y	Y	Y		session	no

For additional system variable information, see these sections:

- [Section 5.2.4, “Using System Variables”](#), discusses the syntax for setting and displaying system variable values.
- [Section 5.2.4.2, “Dynamic System Variables”](#), lists the variables that can be set at runtime.
- Information on tuning system variables can be found in [Section 6.5.2, “Tuning Server Parameters”](#).
- [Section 13.5.4, “InnoDB Startup Options and System Variables”](#), lists [InnoDB](#) system variables.

Note

Some of the following variable descriptions refer to “enabling” or “disabling” a variable. These variables can be enabled with the [SET](#) statement by setting them to [ON](#) or [1](#), or disabled by setting them to [OFF](#) or [0](#). However, to set such a variable on the command line or in an option file, you must set it to [1](#) or [0](#); setting it to [ON](#) or [OFF](#) will not work. For example, on the command line, [--delay_key_write=1](#) works but [--delay_key_write=ON](#) does not.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

- [auto_increment_increment](#)

[auto_increment_increment](#) and [auto_increment_offset](#) are intended for use with master-to-master replication, and can be used to control the operation of [AUTO_INCREMENT](#) columns. Both variables can be set globally or locally, and each can assume an integer value between 1 and 65,535 inclusive. Setting the value of either of these two variables to 0 causes its value to be set to 1 instead. Attempting to set the value of either of these two variables to an integer greater than 65,535 or less than 0 causes its value to be set to 65,535 instead. Attempting to set the value of [auto_increment_increment](#) or

`auto_increment_offset` to a non-integer value gives rise to an error, and the actual value of the variable remains unchanged.

Important

`auto_increment_increment` and `auto_increment_offset` are not intended for use with MySQL Cluster replication. Attempting to set them in a Cluster replication scenario may give rise to unpredictable (and unrecoverable) errors. The use of these variables with Cluster replication is therefore not supported.

These two variables affect `AUTO_INCREMENT` column behavior as follows:

- `auto_increment_increment` controls the interval between successive column values. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc1
-> (col INT NOT NULL AUTO INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.04 sec)

mysql> SET @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 1 |
+-----+-----+
2 rows in set (0.01 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1 |
| 11 |
| 21 |
| 31 |
+-----+
4 rows in set (0.00 sec)
```

(Note how `SHOW VARIABLES` is used here to obtain the current values for these variables.)

- `auto_increment_offset` determines the starting point for the `AUTO_INCREMENT` column value. Consider the following, assuming that these statements are executed during the same session as the example given in the description for `auto_increment_increment`:

```
mysql> SET @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 5 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc2
-> (col INT NOT NULL AUTO INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc2;
+-----+
| col |
+-----+
```

```

 5
15
25
35
+-----+
4 rows in set (0.02 sec)

```

If the value of `auto_increment_offset` is greater than that of `auto_increment_increment`, the value of `auto_increment_offset` is ignored.

Should one or both of these variables be changed and then new rows inserted into a table containing an `AUTO_INCREMENT` column, the results may seem counterintuitive because the series of `AUTO_INCREMENT` values is calculated without regard to any values already present in the column, and the next value inserted is the least value in the series that is greater than the maximum existing value in the `AUTO_INCREMENT` column. In other words, the series is calculated like so:

`auto_increment_offset + N × auto_increment_increment`

where *N* is a positive integer value in the series [1, 2, 3, ...]. For example:

```

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset   | 5 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1 |
| 11 |
| 21 |
| 31 |
+-----+
4 rows in set (0.00 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1 |
| 11 |
| 21 |
| 31 |
| 35 |
| 45 |
| 55 |
| 65 |
+-----+
8 rows in set (0.00 sec)

```

The values shown for `auto_increment_increment` and `auto_increment_offset` generate the series $5 + N \times 10$, that is, [5, 15, 25, 35, 45, ...]. The greatest value present in the `col` column prior to the `INSERT` is 31, and the next available value in the `AUTO_INCREMENT` series is 35, so the inserted values for `col` begin at that point and the results are as shown for the `SELECT` query.

It is important to remember that it is not possible to confine the effects of these two variables to a single table, and thus they do not take the place of the sequences offered by some other database management systems; these variables control the behavior of all `AUTO_INCREMENT` columns in *all* tables on the MySQL server. If one of these variables is set globally, its effects persist until the global value is changed or overridden by setting them locally, or until `mysqld` is restarted. If set locally, the new value affects `AUTO_INCREMENT` columns for all tables into which new rows are inserted by the current user for the duration of the session, unless the values are changed during that session.

The default value of `auto_increment_increment` is 1. See [Auto-Increment in Multiple-Master Replication](#).

`auto_increment_increment` is supported for use with `NDB` tables beginning with MySQL 5.1.20. Previously, setting it when using MySQL Cluster tables or MySQL Cluster Replication produced unpredictable results.

- `auto_increment_offset`

This variable has a default value of 1. For particulars, see the description for `auto_increment_increment`.

`auto_increment_offset` is supported for use with **NDB** tables beginning with MySQL 5.1.20. Previously, setting it when using MySQL Cluster tables or MySQL Cluster Replication produced unpredictable results.

- `automatic_sp_privileges`

When this variable has a value of 1 (the default), the server automatically grants the **EXECUTE** and **ALTER ROUTINE** privileges to the creator of a stored routine, if the user cannot already execute and alter or drop the routine. (The **ALTER ROUTINE** privileges is required to drop the routine.) The server also automatically drops those privileges when the creator drops the routine. If `automatic_sp_privileges` is 0, the server does not automatically add and drop these privileges.

- `back_log`

The number of outstanding connection requests MySQL can have. This comes into play when the main MySQL thread gets very many connection requests in a very short time. It then takes some time (although very little) for the main thread to check the connection and start a new thread. The `back_log` value indicates how many requests can be stacked during this short time before MySQL momentarily stops answering new requests. You need to increase this only if you expect a large number of connections in a short period of time.

In other words, this value is the size of the listen queue for incoming TCP/IP connections. Your operating system has its own limit on the size of this queue. The manual page for the Unix `listen()` system call should have more details. Check your OS documentation for the maximum value for this variable. `back_log` cannot be set higher than your operating system limit.

- `basedir`

The MySQL installation base directory. This variable can be set with the `--basedir` option.

- `binlog_cache_size`

The size of the cache to hold the SQL statements for the binary log during a transaction. A binary log cache is allocated for each client if the server supports any transactional storage engines and if the server has the binary log enabled (`--log-bin` option). If you often use large, multiple-statement transactions, you can increase this cache size to get more performance. The `Binlog_cache_use` and `Binlog_cache_disk_use` status variables can be useful for tuning the size of this variable. See [Section 5.11.4, “The Binary Log”](#).

MySQL Enterprise

For recommendations on the optimum setting for `binlog_cache_size` subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `binlog_format`

The binary logging format, either **STATEMENT**, **ROW**, or **MIXED**. `binlog_format` is set by the `--binlog-format` option at startup, or by the `binlog_format` variable at runtime (you need the **SUPER** privilege to set this variable on a global scope). See [Section 15.1.2, “Replication Formats”](#). The startup variable was added in MySQL 5.1.5, and the runtime variable in MySQL 5.1.8. **MIXED** was added in MySQL 5.1.8.

STATEMENT is used by default. If **MIXED** is specified, statement-based replication is used, too, except for cases where only row-based replication is guaranteed to lead to proper results. For example, this is the case when statements contain user-defined functions (UDF) or the `UUID()` function. An exception to this rule is that **MIXED** always uses statement-based replication for stored functions and triggers.

There are exceptions when you cannot switch the replication format at runtime:

- From within a stored function or a trigger.
 - If **NDB** is enabled.
 - If the session is currently in row-based replication mode and has open temporary tables.
- Trying to switch the format in those cases results in an error.

Before MySQL 5.1.8, switching to row-based replication format would implicitly set `--log-bin-trust-function-creators=1` and `--innodb_locks_unsafe_for_binlog`. MySQL 5.1.8 and later no longer implicitly set these options when row-based replication is used.

- `bulk_insert_buffer_size`

MyISAM uses a special tree-like cache to make bulk inserts faster for `INSERT ... SELECT`, `INSERT ... VALUES (...), (...), ...`, and `LOAD DATA INFILE` when adding data to non-empty tables. This variable limits the size of the cache tree in bytes per thread. Setting it to 0 disables this optimization. The default value is 8MB.

- `character_set_client`

The character set for statements that arrive from the client.

- `character_set_connection`

The character set used for literals that do not have a character set introducer and for number-to-string conversion.

- `character_set_database`

The character set used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `character_set_server`.

- `character_set_filesystem`

The filesystem character set. This variable is used to interpret string literals that refer to filenames, such as in the `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function. Such filenames are converted from `character_set_client` to `character_set_filesystem` before the file opening attempt occurs. The default value is `binary`, which means that no conversion occurs. For systems on which multi-byte filenames are allowed, a different value may be more appropriate. For example, if the system represents filenames using UTF-8, set `character_set_filesystem` to `'utf8'`. This variable was added in MySQL 5.1.6.

- `character_set_results`

The character set used for returning query results to the client.

- `character_set_server`

The server's default character set.

- `character_set_system`

The character set used by the server for storing identifiers. The value is always `utf8`.

- `character_sets_dir`

The directory where character sets are installed.

- `collation_connection`

The collation of the connection character set.

- `collation_database`

The collation used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `collation_server`.

- `collation_server`

The server's default collation.

- `completion_type`

The transaction completion type:

- If the value is 0 (the default), `COMMIT` and `ROLLBACK` are unaffected.
- If the value is 1, `COMMIT` and `ROLLBACK` are equivalent to `COMMIT AND CHAIN` and `ROLLBACK AND CHAIN`, respectively. (A new transaction starts immediately with the same isolation level as the just-terminated transaction.)
- If the value is 2, `COMMIT` and `ROLLBACK` are equivalent to `COMMIT RELEASE` and `ROLLBACK RELEASE`, respectively.

(The server disconnects after terminating the transaction.)

- `concurrent_insert`

If 1 (the default), MySQL allows `INSERT` and `SELECT` statements to run concurrently for `MyISAM` tables that have no free blocks in the middle of the data file. You can turn this option off by starting `mysqld` with `--safe` or `--skip-new`.

This variable can take three integer values:

Value	Description
0	Off
1	(Default) Enables concurrent insert for <code>MyISAM</code> tables that don't have holes
2	Enables concurrent inserts for all <code>MyISAM</code> tables, even those that have holes. For a table with a hole, new rows are inserted at the end of the table if it is in use by another thread. Otherwise, MySQL acquires a normal write lock and inserts the row into the hole.

See also [Section 6.3.3, “Concurrent Inserts”](#).

- `connect_timeout`

The number of seconds that the `mysqld` server waits for a connect packet before responding with `Bad handshake`.

- `datadir`

The MySQL data directory. This variable can be set with the `--datadir` option.

- `date_format`

This variable is not implemented.

- `datetime_format`

This variable is not implemented.

- `default_week_format`

The default mode value to use for the `WEEK()` function. See [Section 11.6, “Date and Time Functions”](#).

- `delay_key_write`

This option applies only to `MyISAM` tables. It can have one of the following values to affect handling of the `DELAY_KEY_WRITE` table option that can be used in `CREATE TABLE` statements.

Option	Description
<code>OFF</code>	<code>DELAY_KEY_WRITE</code> is ignored.
<code>ON</code>	MySQL honors any <code>DELAY_KEY_WRITE</code> option specified in <code>CREATE TABLE</code> statements. This is the default value.
<code>ALL</code>	All new opened tables are treated as if they were created with the <code>DELAY_KEY_WRITE</code> option enabled.

If `DELAY_KEY_WRITE` is enabled for a table, the key buffer is not flushed for the table on every index update, but only when the table is closed. This speeds up writes on keys a lot, but if you use this feature, you should add automatic checking of all `MyISAM` tables by starting the server with the `--myisam-recover` option (for example, `--myisam-recover=BACKUP, FORCE`). See [Section 5.2.2, “Command Options”](#), and [Section 13.4.1, “MyISAM Startup Options”](#).

Note that if you enable external locking with `--external-locking`, there is no protection against index corruption for tables that use delayed key writes.

- `delayed_insert_limit`

After inserting `delayed_insert_limit` delayed rows, the `INSERT DELAYED` handler thread checks whether there are any `SELECT` statements pending. If so, it allows them to execute before continuing to insert delayed rows.

- `delayed_insert_timeout`

How many seconds an `INSERT DELAYED` handler thread should wait for `INSERT` statements before terminating.

- `delayed_queue_size`

This is a per-table limit on the number of rows to queue when handling `INSERT DELAYED` statements. If the queue becomes full, any client that issues an `INSERT DELAYED` statement waits until there is room in the queue again.

- `div_precision_increment`

This variable indicates the number of digits of precision by which to increase the result of division operations performed with the `/` operator. The default value is 4. The minimum and maximum values are 0 and 30, respectively. The following example illustrates the effect of increasing the default value.

```
mysql> SELECT 1/7;
+-----+
| 1/7    |
+-----+
| 0.1429 |
+-----+
mysql> SET div_precision_increment = 12;
mysql> SELECT 1/7;
+-----+
| 1/7    |
+-----+
| 0.142857142857 |
+-----+
```

- `event_scheduler`

This variable indicates the status of the Event Scheduler; as of MySQL 5.1.12, possible values are `ON`, `OFF`, and `DISABLED`, with the default being `OFF`. This variable and its effects on the Event Scheduler's operation are discussed in greater detail in the [Overview section of the Events chapter](#).

This variable was added in MySQL 5.1.6.

- `engine_condition_pushdown`

This variable applies to NDB. By default it is 0 (`OFF`): If you execute a query such as `SELECT * FROM t WHERE mycol = 42`, where `mycol` is a non-indexed column, the query is executed as a full table scan on every NDB node. Each node sends every row to the MySQL server, which applies the `WHERE` condition. If `engine_condition_pushdown` is set to 1 (`ON`), the condition is “pushed down” to the storage engine and sent to the NDB nodes. Each node uses the condition to perform the scan, and only sends back to the MySQL server the rows that match the condition.

- `expire_logs_days`

The number of days for automatic binary log removal. The default is 0, which means “no automatic removal.” Possible removals happen at startup and at binary log rotation.

- `flush`

If `ON`, the server flushes (synchronizes) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See [Section B.1.4.2, “What to Do If MySQL Keeps Crashing”](#). This variable is set to `ON` if you start `mysqld` with the `--flush` option.

- `flush_time`

If this is set to a non-zero value, all tables are closed every `flush_time` seconds to free up resources and synchronize unflushed data to disk. We recommend that this option be used only on Windows 9x or Me, or on systems with minimal resources.

- `ft_boolean_syntax`

The list of operators supported by boolean full-text searches performed using `IN BOOLEAN MODE`. See [Section 11.8.1, “Boolean Full-Text Searches”](#).

The default variable value is `'+ -><()~*:""&| '`. The rules for changing the value are as follows:

- Operator function is determined by position within the string.

- The replacement value must be 14 characters.
- Each character must be an ASCII non-alphanumeric character.
- Either the first or second character must be a space.
- No duplicates are allowed except the phrase quoting operators in positions 11 and 12. These two characters are not required to be the same, but they are the only two that may be.
- Positions 10, 13, and 14 (which by default are set to ':', '&', and '|') are reserved for future extensions.

- `ft_max_word_len`

The maximum length of the word to be included in a `FULLTEXT` index.

Note

`FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_min_word_len`

The minimum length of the word to be included in a `FULLTEXT` index.

Note

`FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_query_expansion_limit`

The number of top matches to use for full-text searches performed using `WITH QUERY EXPANSION`.

- `ft_stopword_file`

The file from which to read the list of stopwords for full-text searches. All the words from the file are used; comments are *not* honored. By default, a built-in list of stopwords is used (as defined in the `storage/myisam/ft_static.c` file). Setting this variable to the empty string (`' '`) disables stopwords filtering.

Note

`FULLTEXT` indexes must be rebuilt after changing this variable or the contents of the stopwords file. Use `REPAIR TABLE tbl_name QUICK`.

- `general_log`

Whether the general query log is enabled. The value can be 0 (or `OFF`) to disable the log or 1 (or `ON`) to enable the log. The default value depends on whether the `--log` option is given. The destination for log output is controlled by the `log_output` system variable; if that value is `NONE`, no log entries are written even if the log is enabled. The `general_log` variable was added in MySQL 5.1.12.

- `general_log_file`

The name of the general query log file. The default value is `host_name.log`, but the initial value can be changed with the `--log` option. This variable was added in MySQL 5.1.12.

- `group_concat_max_len`

The maximum allowed result length for the `GROUP_CONCAT()` function. The default is 1024.

- `have_archive`

`YES` if `mysqld` supports `ARCHIVE` tables, `NO` if not.

- `have_blackhole_engine`

`YES` if `mysqld` supports `BLACKHOLE` tables, `NO` if not.

- `have_compress`

YES if the `zlib` compression library is available to the server, NO if not. If not, the `COMPRESS()` and `UNCOMPRESS()` functions cannot be used.

- `have_crypt`

YES if the `crypt()` system call is available to the server, NO if not. If not, the `ENCRYPT()` function cannot be used.

- `have_csv`

YES if `mysqld` supports `ARCHIVE` tables, NO if not.

- `have_dynamic_loading`

YES if `mysqld` supports dynamic loading of plugins, NO if not. This variable was added in MySQL 5.1.10.

- `have_example_engine`

YES if `mysqld` supports `EXAMPLE` tables, NO if not.

- `have_federated_engine`

YES if `mysqld` supports `FEDERATED` tables, NO if not.

- `have_geometry`

YES if the server supports spatial data types, NO if not.

- `have_innodb`

YES if `mysqld` supports `InnoDB` tables. `DISABLED` if `--skip-innodb` is used.

- `have_ndbcluster`

YES if `mysqld` supports `NDB Cluster` tables. `DISABLED` if `--skip-ndbcluster` is used.

- `have_partitioning`

YES if `mysqld` supports partitioning. Added in MySQL 5.1.1 as `have_partition_engine` and renamed to `have_partitioning` in 5.1.6.

- `have_openssl`

YES if `mysqld` supports SSL connections, NO if not.

- `have_query_cache`

YES if `mysqld` supports the query cache, NO if not.

- `have_row_based_replication`

YES if the server can perform replication using row-based binary logging. If the value is NO, the server can use only statement-based logging. See [Section 15.1.2, “Replication Formats”](#). This variable was added in MySQL 5.1.5. This variable was removed in 5.1.15.

- `have_rtree_keys`

YES if `RTREE` indexes are available, NO if not. (These are used for spatial indexes in `MyISAM` tables.)

- `have_symlink`

YES if symbolic link support is enabled, NO if not. This is required on Unix for support of the `DATA DIRECTORY` and `INDEX DIRECTORY` table options, and on Windows for support of data directory symlinks.

- `hostname`

The server sets this variable to the server hostname at startup. This variable was added in MySQL 5.1.17.

- `init_connect`

A string to be executed by the server for each client that connects. The string consists of one or more SQL statements. To specify multiple statements, separate them by semicolon characters. For example, each client begins by default with autocommit mode enabled. There is no global system variable to specify that autocommit should be disabled by default, but `init_connect` can be used to achieve the same effect:

```
SET GLOBAL init_connect='SET AUTOCOMMIT=0';
```

This variable can also be set on the command line or in an option file. To set the variable as just shown using an option file, include these lines:

```
[mysqld]  
init_connect='SET AUTOCOMMIT=0'
```

Note that the content of `init_connect` is not executed for users that have the `SUPER` privilege. This is done so that an erroneous value for `init_connect` does not prevent all clients from connecting. For example, the value might contain a statement that has a syntax error, thus causing client connections to fail. Not executing `init_connect` for users that have the `SUPER` privilege enables them to open a connection and fix the `init_connect` value.

- `init_file`

The name of the file specified with the `--init-file` option when you start the server. This should be a file containing SQL statements that you want the server to execute when it starts. Each statement must be on a single line and should not include comments.

Note that the `--init-file` option is unavailable if MySQL was configured with the `--disable-grant-options` option. See [Section 2.9.2, “Typical configure Options”](#).

- `init_slave`

This variable is similar to `init_connect`, but is a string to be executed by a slave server each time the SQL thread starts. The format of the string is the same as for the `init_connect` variable.

- `innodb_xxx`

InnoDB system variables are listed in [Section 13.5.4, “InnoDB Startup Options and System Variables”](#).

- `interactive_timeout`

The number of seconds the server waits for activity on an interactive connection before closing it. An interactive client is defined as a client that uses the `CLIENT_INTERACTIVE` option to `mysql_real_connect()`. See also `wait_timeout`.

- `join_buffer_size`

The size of the buffer that is used for joins that do not use indexes and thus perform full table scans. Normally, the best way to get fast joins is to add indexes. Increase the value of `join_buffer_size` to get a faster full join when adding indexes is not possible. One join buffer is allocated for each full join between two tables. For a complex join between several tables for which indexes are not used, multiple join buffers might be necessary.

- `keep_files_on_create`

If a `MyISAM` table is created with no `DATA DIRECTORY` option, the `.MYD` file is created in the database directory. By default, if `MyISAM` finds an existing `.MYD` file in this case, it overwrites it. The same applies to `.MYI` files for tables created with no `INDEX DIRECTORY` option. To suppress this behavior, set the `keep_files_on_create` variable to `ON` (1), in which case `MyISAM` will not overwrite existing files and returns an error instead. The default value is `OFF` (0).

If a `MyISAM` table is created with a `DATA DIRECTORY` or `INDEX DIRECTORY` option and an existing `.MYD` or `.MYI` file is found, `MyISAM` always returns an error. It will not overwrite a file in the specified directory.

This variable was added in MySQL 5.1.23.

- `key_buffer_size`

Index blocks for **MyISAM** tables are buffered and are shared by all threads. **key_buffer_size** is the size of the buffer used for index blocks. The key buffer is also known as the key cache.

The maximum allowable setting for **key_buffer_size** is 4GB. The effective maximum size might be less, depending on your available physical RAM and per-process RAM limits imposed by your operating system or hardware platform.

Increase the value to get better index handling (for all reads and multiple writes) to as much as you can afford. Using a value that is 25% of total memory on a machine that mainly runs MySQL is quite common. However, if you make the value too large (for example, more than 50% of your total memory) your system might start to page and become extremely slow. MySQL relies on the operating system to perform filesystem caching for data reads, so you must leave some room for the filesystem cache. Consider also the memory requirements of other storage engines.

For even more speed when writing many rows at the same time, use **LOCK TABLES**. See [Section 6.2.17, “Speed of INSERT Statements”](#).

You can check the performance of the key buffer by issuing a **SHOW STATUS** statement and examining the **Key_read_requests**, **Key_reads**, **Key_write_requests**, and **Key_writes** status variables. (See [Section 12.5.4, “SHOW Syntax”](#).) The **Key_reads/Key_read_requests** ratio should normally be less than 0.01. The **Key_writes/Key_write_requests** ratio is usually near 1 if you are using mostly updates and deletes, but might be much smaller if you tend to do updates that affect many rows at the same time or if you are using the **DELAY_KEY_WRITE** table option.

The fraction of the key buffer in use can be determined using **key_buffer_size** in conjunction with the **Key_blocks_unused** status variable and the buffer block size, which is available from the **key_cache_block_size** system variable:

```
1 - ((Key_blocks_unused × key_cache_block_size) / key_buffer_size)
```

This value is an approximation because some space in the key buffer may be allocated internally for administrative structures.

It is possible to create multiple **MyISAM** key caches. The size limit of 4GB applies to each cache individually, not as a group. See [Section 6.4.6, “The MyISAM Key Cache”](#).

- **key_cache_age_threshold**

Name	key_cache_age_threshold	
Description	This characterizes the number of hits a hot block has to be untouched until it is considered aged enough to be downgraded to a warm block. This specifies the percentage ratio of that number of hits to the total number of blocks in key cache	
Option Sets Variable	Yes, key_cache_age_threshold	
Variable Name	key_cache_age_threshold	
Variable Scope	Server	
Dynamic Variable	no	
Value Set	Type	numeric
	Default	300
	Min Value	100

This value controls the demotion of buffers from the hot sub-chain of a key cache to the warm sub-chain. Lower values cause demotion to happen more quickly. The minimum value is 100. The default value is 300. See [Section 6.4.6, “The MyISAM Key Cache”](#).

- **key_cache_block_size**

The size in bytes of blocks in the key cache. The default value is 1024. See [Section 6.4.6, “The MyISAM Key Cache”](#).

- **key_cache_division_limit**

The division point between the hot and warm sub-chains of the key cache buffer chain. The value is the percentage of the buffer chain to use for the warm sub-chain. Allowable values range from 1 to 100. The default value is 100. See [Section 6.4.6, “The MyISAM Key Cache”](#).

- `language`

The language used for error messages.

- `large_file_support`

Whether `mysqld` was compiled with options for large file support.

- `large_pages`

Whether large page support is enabled.

- `lc_time_names`

This variable specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()` and `MONTHNAME()` functions. Locale names are POSIX-style values such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting. For further information, see [Section 5.10.9, “MySQL Server Locale Support”](#). This variable was added in MySQL 5.1.12.

- `license`

The type of license the server has.

- `local_infile`

Whether `LOCAL` is supported for `LOAD DATA INFILE` statements. See [Section 5.6.4, “Security Issues with LOAD DATA LOCAL”](#).

- `locked_in_memory`

Whether `mysqld` was locked in memory with `--memlock`.

- `log`

Whether logging of all statements to the general query log is enabled. See [Section 5.11.3, “The General Query Log”](#).

- `log_bin`

Whether the binary log is enabled. See [Section 5.11.4, “The Binary Log”](#).

- `log_bin_trust_function_creators`

This variable applies when binary logging is enabled. It controls whether stored function creators can be trusted not to create stored functions that will cause unsafe events to be written to the binary log. If set to 0 (the default), users are not allowed to create or alter stored routines unless they have the `SUPER` privilege in addition to the `CREATE ROUTINE` or `ALTER ROUTINE` privilege. A setting of 0 also enforces the restriction that a function must be declared with the `DETERMINISTIC` characteristic, or with the `READS SQL DATA` or `NO SQL` characteristic. If the variable is set to 1, MySQL does not enforce these restrictions on stored function creation. See [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

- `log_error`

The location of the error log.

- `log_output`

The destination for general query log and slow query log output. The value can be a comma-separated list of one or more of the words `TABLE` (log to tables), `FILE` (log to files), or `NONE` (do not log to tables or files). The default value is `TABLE`. `NONE`, if present, takes precedence over any other specifiers. If the value is `NONE` log entries are not written even if the logs are enabled. If the logs are not enabled, no logging occurs even if the value of `log_output` is not `NONE`. For more information, see [Section 5.11.1, “Selecting General Query and Slow Query Log Output Destinations”](#). This variable was added in MySQL 5.1.6.

- `log_queries_not_using_indexes`

Whether queries that do not use indexes are logged to the slow query log. See [Section 5.11.5, “The Slow Query Log”](#). This variable was added in MySQL 5.1.11.

- `log_slave_updates`

Whether updates received by a slave server from a master server should be logged to the slave's own binary log. Binary logging must be enabled on the slave for this variable to have any effect. See [Section 15.1.3, “Replication Options and Variables”](#).

- `log_slow_queries`

Whether slow queries should be logged. “Slow” is determined by the value of the `long_query_time` variable. See [Section 5.11.5, “The Slow Query Log”](#).

- `log_warnings`

Whether to produce additional warning messages. It is enabled (1) by default and can be disabled by setting it to 0. Aborted connections are not logged to the error log unless the value is greater than 1.

- `long_query_time`

If a query takes longer than this many seconds, the server increments the `Slow_queries` status variable. If you are using the `-log-slow-queries` option, the query is logged to the slow query log file. This value is measured in real time, not CPU time, so a query that is under the threshold on a lightly loaded system might be above the threshold on a heavily loaded one. The minimum value is 1. The default is 10. See [Section 5.11.5, “The Slow Query Log”](#).

- `low_priority_updates`

If set to 1, all `INSERT`, `UPDATE`, `DELETE`, and `LOCK TABLE WRITE` statements wait until there is no pending `SELECT` or `LOCK TABLE READ` on the affected table. This affects only storage engines that use only table-level locking (`MyISAM`, `MEMORY`, `MERGE`). This variable previously was named `sql_low_priority_updates`.

- `lower_case_file_system`

This variable describes the case sensitivity of filenames on the filesystem where the data directory is located. `OFF` means filenames are case sensitive, `ON` means they are not case sensitive.

- `lower_case_table_names`

If set to 1, table names are stored in lowercase on disk and table name comparisons are not case sensitive. If set to 2 table names are stored as given but compared in lowercase. This option also applies to database names and table aliases. See [Section 8.2.2, “Identifier Case Sensitivity”](#).

If you are using `InnoDB` tables, you should set this variable to 1 on all platforms to force names to be converted to lowercase.

You should *not* set this variable to 0 if you are running MySQL on a system that does not have case-sensitive filenames (such as Windows or Mac OS X). If this variable is not set at startup and the filesystem on which the data directory is located does not have case-sensitive filenames, MySQL automatically sets `lower_case_table_names` to 2.

- `max_allowed_packet`

The maximum size of one packet or any generated/intermediate string.

The packet message buffer is initialized to `net_buffer_length` bytes, but can grow up to `max_allowed_packet` bytes when needed. This value by default is small, to catch large (possibly incorrect) packets.

You must increase this value if you are using large `BLOB` columns or long strings. It should be as big as the largest `BLOB` you want to use. The protocol limit for `max_allowed_packet` is 1GB.

- `max_binlog_cache_size`

If a multiple-statement transaction requires more than this many bytes of memory, the server generates a `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage` error. The minimum value is 4096, the maximum and default values are 4GB.

- `max_binlog_size`

If a write to the binary log causes the current log file size to exceed the value of this variable, the server rotates the binary logs (closes the current file and opens the next one). You cannot set this variable to more than 1GB or to less than 4096 bytes. The default value is 1GB.

A transaction is written in one chunk to the binary log, so it is never split between several binary logs. Therefore, if you have big transactions, you might see binary logs larger than `max_binlog_size`.

If `max_relay_log_size` is 0, the value of `max_binlog_size` applies to relay logs as well.

- `max_connect_errors`

If there are more than this number of interrupted connections from a host, that host is blocked from further connections. You can unblock blocked hosts with the `FLUSH HOSTS` statement.

- `max_connections`

The number of simultaneous client connections allowed. By default, this is 150, beginning with MySQL 5.1.15. (Previously, the default was 100.) See [Section B.1.2.6, “Too many connections”](#), for more information.

MySQL Enterprise

For notification that the maximum number of connections is getting dangerously high and for advice on setting the optimum value for `max_connections` subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

Increasing this value increases the number of file descriptors that `mysqld` requires. See [Section 6.4.8, “How MySQL Opens and Closes Tables”](#), for comments on file descriptor limits.

- `max_delayed_threads`

Do not start more than this number of threads to handle `INSERT DELAYED` statements. If you try to insert data into a new table after all `INSERT DELAYED` threads are in use, the row is inserted as if the `DELAYED` attribute wasn't specified. If you set this to 0, MySQL never creates a thread to handle `DELAYED` rows; in effect, this disables `DELAYED` entirely.

- `max_error_count`

The maximum number of error, warning, and note messages to be stored for display by the `SHOW ERRORS` and `SHOW WARNINGS` statements.

- `max_heap_table_size`

This variable sets the maximum size to which `MEMORY` tables are allowed to grow. The value of the variable is used to calculate `MEMORY` table `MAX_ROWS` values. Setting this variable has no effect on any existing `MEMORY` table, unless the table is re-created with a statement such as `CREATE TABLE` or altered with `ALTER TABLE` or `TRUNCATE TABLE`.

MySQL Enterprise

Subscribers to the MySQL Enterprise Monitor receive recommendations for the optimum setting for `max_heap_table_size`. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `max_insert_delayed_threads`

This variable is a synonym for `max_delayed_threads`.

- `max_join_size`

Do not allow `SELECT` statements that probably need to examine more than `max_join_size` rows (for single-table statements) or row combinations (for multiple-table statements) or that are likely to do more than `max_join_size` disk seeks. By setting this value, you can catch `SELECT` statements where keys are not used properly and that would probably take a long time. Set it if your users tend to perform joins that lack a `WHERE` clause, that take a long time, or that return millions of rows.

Setting this variable to a value other than `DEFAULT` resets the value of `SQL_BIG_SELECTS` to 0. If you set the `SQL_BIG_SELECTS` value again, the `max_join_size` variable is ignored.

If a query result is in the query cache, no result size check is performed, because the result has previously been computed and it does not burden the server to send it to the client.

This variable previously was named `sql_max_join_size`.

- `max_length_for_sort_data`

The cutoff on the size of index values that determines which `filesort` algorithm to use. See [Section 6.2.11, “ORDER BY Optimization”](#).

- `max_prepared_statement_count`

This variable limits the total number of prepared statements in the server. It can be used in environments where there is the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. The default value is 16382. The allowable range of values is from 0 to 1 million. If the value is set lower than the current number of prepared statements, existing statements are not affected and can be used, but no new statements can be prepared until the current number drops below the limit. This variable was added in MySQL 5.1.10.

- `max_relay_log_size`

If a write by a replication slave to its relay log causes the current log file size to exceed the value of this variable, the slave rotates the relay logs (closes the current file and opens the next one). If `max_relay_log_size` is 0, the server uses `max_binlog_size` for both the binary log and the relay log. If `max_relay_log_size` is greater than 0, it constrains the size of the relay log, which enables you to have different sizes for the two logs. You must set `max_relay_log_size` to between 4096 bytes and 1GB (inclusive), or to 0. The default value is 0. See [Section 15.4.1, “Replication Implementation Details”](#).

- `max_seeks_for_key`

Limit the assumed maximum number of seeks when looking up rows based on a key. The MySQL optimizer assumes that no more than this number of key seeks are required when searching for matching rows in a table by scanning an index, regardless of the actual cardinality of the index (see [Section 12.5.4.18, “SHOW INDEX Syntax”](#)). By setting this to a low value (say, 100), you can force MySQL to prefer indexes instead of table scans.

- `max_sort_length`

The number of bytes to use when sorting `BLOB` or `TEXT` values. Only the first `max_sort_length` bytes of each value are used; the rest are ignored.

- `max_sp_recursion_depth`

The number of times that a stored procedure may call itself. The default value for this option is 0, which completely disallows recursion in stored procedures. The maximum value is 255.

This variable can be set globally and per session.

- `max_tmp_tables`

The maximum number of temporary tables a client can keep open at the same time. (This option does not yet do anything.)

- `max_user_connections`

The maximum number of simultaneous connections allowed to any given MySQL account. A value of 0 means “no limit.”

This variable has both a global scope and a (read-only) session scope. The session variable has the same value as the global variable unless the current account has a non-zero `MAX_USER_CONNECTIONS` resource limit. In that case, the session value reflects the account limit.

- `max_write_lock_count`

After this many write locks, allow some pending read lock requests to be processed in between.

- `multi_range_count`

The maximum number of ranges to send to a table handler at once during range selects. The default value is 256. Sending multiple ranges to a handler at once can improve the performance of certain selects dramatically. This especially true for the NDB Cluster table handler, which needs to send the range requests to all nodes. Sending a batch of those requests at once reduces the communication costs significantly.

- `myisam_block_size`

The block size to be used for `MyISAM` index pages.

- `myisam_data_pointer_size`

The default pointer size in bytes, to be used by `CREATE TABLE` for `MyISAM` tables when no `MAX_ROWS` option is specified. This variable cannot be less than 2 or larger than 7. The default value is 6. See [Section B.1.2.11, “The table is full”](#).

- `myisam_max_extra_sort_file_size` (*DEPRECATED*)

Note

This variable is not supported in MySQL 5.1. See *MySQL 5.0 Reference Manual* for more information.

- `myisam_max_sort_file_size`

The maximum size of the temporary file that MySQL is allowed to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

The default value is 2GB. If `MyISAM` index files exceed this size and disk space is available, increasing the value may help performance.

- `myisam_recover_options`

The value of the `--myisam-recover` option. See [Section 5.2.2, “Command Options”](#).

- `myisam_repair_threads`

If this value is greater than 1, `MyISAM` table indexes are created in parallel (each index in its own thread) during the `Repair by sorting` process. The default value is 1.

Note

Multi-threaded repair is still *beta-quality* code.

- `myisam_sort_buffer_size`

The size of the buffer that is allocated when sorting `MyISAM` indexes during a `REPAIR TABLE` or when creating indexes with `CREATE INDEX` or `ALTER TABLE`.

- `myisam_stats_method`

How the server treats `NULL` values when collecting statistics about the distribution of index values for `MyISAM` tables. This variable has two possible values, `nulls_equal` and `nulls_unequal`. For `nulls_equal`, all `NULL` index values are considered equal and form a single value group that has a size equal to the number of `NULL` values. For `nulls_unequal`, `NULL` values are considered unequal, and each `NULL` forms a distinct value group of size 1.

The method that is used for generating table statistics influences how the optimizer chooses indexes for query execution, as described in [Section 6.4.7, “MyISAM Index Statistics Collection”](#).

- `myisam_use_mmap`

Use memory mapping for reading and writing `MyISAM` tables. This variable was added in MySQL 5.1.4.

- `multi_read_range`

Specifies the maximum number of ranges to send to a storage engine during range selects. The default value is 256. Sending multiple ranges to an engine is a feature that can improve the performance of certain selects dramatically, particularly for `NDBCLUSTER`. This engine needs to send the range requests to all nodes, and sending many of those requests at once reduces the communication costs significantly.

- `named_pipe`

(Windows only.) Indicates whether the server supports connections over named pipes.

- `ndb_autoincrement_prefetch_sz`

Determines the probability of gaps in an autoincremented column. Set to 1 to minimize this. Set to a high value for optimization —

makes inserts faster, but decreases the likelihood that consecutive autoincrement numbers will be used in a batch of inserts. Default value: **32**. Minimum value: **1**.

- **ndb_cache_check_time**

The number of milliseconds to wait before checking the **NDB** query cache. Setting this to **0** (the default and minimum value) means that the **NDB** query cache will be checked for validation on every query.

The recommended maximum value for this variable is **1000**, which means that the query cache is checked once per second. A larger value means the **NDB** query cache is less often checked and invalidated due to updates on a different **mysql**. It is generally not desirable to set this to a value greater than **2000**.

- **ndb_extra_logging**

This variable can be set to a non-zero value to enable extra **NDB** logging for debugging or troubleshooting purposes. The default value is **0**.

This variable was added in MySQL 5.1.6.

- **ndb_force_send**

Forces sending of buffers to **NDB** immediately, without waiting for other threads. Defaults to **ON**.

- **ndb_index_stat_cache_entries**

Sets the granularity of the statistics by determining the number of starting and ending keys to store in the statistics memory cache. Zero means no caching takes place; in this case, the data nodes are always queried directly. Default value: **32**.

- **ndb_index_stat_enable**

Use **NDB** index statistics in query optimization. Defaults to **ON**.

- **ndb_index_stat_update_freq**

How often to query data nodes instead of the statistics cache. For example, a value of **20** (the default) means to direct every 20th query to the data nodes.

- **ndb_optimized_node_selection**

Causes an SQL node to contact the nearest data node in the cluster. Enabled by default. Set to **0** or **OFF** to disable, in which case the SQL node attempts to contact data nodes in succession.

Added in MySQL 4.1.9.

- **ndb_report_thresh_binlog_epoch_slip**

This is a threshold on the number of epochs to be behind before reporting binlog status. For example, a value of **3** (the default) means that if the difference between which epoch has been received from the storage nodes and which epoch has been applied to the binlog is 3 or more, a status message will be sent to the cluster log.

- **ndb_report_thresh_binlog_mem_usage**

This is a threshold on the percentage of free memory remaining before reporting binlog status. For example, a value of **10** (the default) means that if the amount of available memory for receiving binlog data from the data nodes falls below 10%, a status message will be sent to the cluster log.

- **ndb_use_copying_alter_table**

Forces **NDB** to use copying of tables in the event of problems with online **ALTER TABLE** operations. The default value is **OFF**.

This variable was added in MySQL 5.1.12.

- **ndb_use_exact_count**

Forces **NDB** to use a count of records during **SELECT COUNT(*)** query planning to speed up this type of query. The default value is **ON**. For faster queries overall, disable this feature by setting the value of **ndb_use_exact_count** to **OFF**.

- `ndb_use_transactions`

You can disable NDB transaction support by setting this variable's values to `OFF` (not recommended). The default is `ON`.

- `ndb_wait_connected`

MySQL Cluster 5.1 Carrier Grade Edition

This variable is available in MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

This variable can be used to cause the MySQL server to wait a given period of time for connections to MySQL Cluster management and data nodes to be established before accepting MySQL client connections. The time is specified in seconds. The default value is 0.

- `net_buffer_length`

Each client thread is associated with a connection buffer and result buffer. Both begin with a size given by `net_buffer_length` but are dynamically enlarged up to `max_allowed_packet` bytes as needed. The result buffer shrinks to `net_buffer_length` after each SQL statement.

This variable should not normally be changed, but if you have very little memory, you can set it to the expected length of statements sent by clients. If statements exceed this length, the connection buffer is automatically enlarged. The maximum value to which `net_buffer_length` can be set is 1MB.

- `net_read_timeout`

The number of seconds to wait for more data from a connection before aborting the read. This timeout applies only to TCP/IP connections, not to connections made via Unix socket files, named pipes, or shared memory. When the server is reading from the client, `net_read_timeout` is the timeout value controlling when to abort. When the server is writing to the client, `net_write_timeout` is the timeout value controlling when to abort. See also `slave_net_timeout`.

- `net_retry_count`

If a read on a communication port is interrupted, retry this many times before giving up. This value should be set quite high on FreeBSD because internal interrupts are sent to all threads.

- `net_write_timeout`

The number of seconds to wait for a block to be written to a connection before aborting the write. This timeout applies only to TCP/IP connections, not to connections made via Unix socket files, named pipes, or shared memory. See also `net_read_timeout`.

- `new`

This variable was used in MySQL 4.0 to turn on some 4.1 behaviors, and is retained for backward compatibility. In MySQL 5.1, its value is always `OFF`.

- `old`

`old` is a compatibility variable. It is disabled by default, but can be enabled at startup to revert the server to behaviors present in older versions.

Currently, when `old` is enabled, it changes the default scope of index hints to that used prior to MySQL 5.1.17. That is, index hints with no `FOR` clause apply only to how indexes are used for row retrieval and not to resolution of `ORDER BY` or `GROUP BY` clauses. (See Section 12.2.7.2, “Index Hint Syntax”.) Take care about enabling this in a replication setup. With statement-based binary logging, having different modes for the master and slaves might lead to replication errors.

This variable was added as `old_mode` in MySQL 5.1.17 and renamed to `old` in MySQL 5.1.18.

- `old_passwords`

Whether the server should use pre-4.1-style passwords for MySQL user accounts. See Section B.1.2.3, “Client does not support authentication protocol”.

- `one_shot`

This is not a variable, but it can be used when setting some variables. It is described in Section 12.5.3, “SET Syntax”.

- `open_files_limit`

The number of files that the operating system allows `mysqld` to open. This is the real value allowed by the system and might be different from the value you gave using the `--open-files-limit` option to `mysqld` or `mysqld_safe`. The value is 0 on systems where MySQL can't change the number of open files.

- `optimizer_prune_level`

Controls the heuristics applied during query optimization to prune less-promising partial plans from the optimizer search space. A value of 0 disables heuristics so that the optimizer performs an exhaustive search. A value of 1 causes the optimizer to prune plans based on the number of rows retrieved by intermediate plans.

- `optimizer_search_depth`

The maximum depth of search performed by the query optimizer. Values larger than the number of relations in a query result in better query plans, but take longer to generate an execution plan for a query. Values smaller than the number of relations in a query return an execution plan quicker, but the resulting plan may be far from being optimal. If set to 0, the system automatically picks a reasonable value. If set to the maximum number of tables used in a query plus 2, the optimizer switches to the algorithm used in MySQL 5.0.0 (and previous versions) for performing searches.

- `pid_file`

The pathname of the process ID (PID) file. This variable can be set with the `--pid-file` option.

- `plugin_dir`

The pathname of the plugins directory. This variable was added in MySQL 5.1.2.

- `port`

The number of the port on which the server listens for TCP/IP connections. This variable can be set with the `--port` option.

- `preload_buffer_size`

The size of the buffer that is allocated when preloading indexes.

- `prepared_STATEMENT_count`

The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_STATEMENT_count` system variable.) This variable was added in MySQL 5.1.10. In MySQL 5.1.14, it was converted to the global `Prepared_STATEMENT_count` status variable.

- `protocol_version`

The version of the client/server protocol used by the MySQL server.

- `query_alloc_block_size`

The allocation size of memory blocks that are allocated for objects created during statement parsing and execution. If you have problems with memory fragmentation, it might help to increase this a bit.

- `query_cache_limit`

Don't cache results that are larger than this number of bytes. The default value is 1MB.

- `query_cache_min_res_unit`

The minimum size (in bytes) for blocks allocated by the query cache. The default value is 4096 (4KB). Tuning information for this variable is given in [Section 6.5.4.3, “Query Cache Configuration”](#).

- `query_cache_size`

The amount of memory allocated for caching query results. The default value is 0, which disables the query cache. The allowable values are multiples of 1024; other values are rounded down to the nearest multiple. Note that `query_cache_size` bytes of memory are allocated even if `query_cache_type` is set to 0. See [Section 6.5.4.3, “Query Cache Configuration”](#), for more information.

- `query_cache_type`

Set the query cache type. Setting the `GLOBAL` value sets the type for all clients that connect thereafter. Individual clients can set the `SESSION` value to affect their own use of the query cache. Possible values are shown in the following table:

Option	Description
0 or <code>OFF</code>	Don't cache results in or retrieve results from the query cache. Note that this does not deallocate the query cache buffer. To do that, you should set <code>query_cache_size</code> to 0.
1 or <code>ON</code>	Cache all query results except for those that begin with <code>SELECT SQL_NO_CACHE</code> .
2 or <code>DEMAND</code>	Cache results only for queries that begin with <code>SELECT SQL_CACHE</code> .

This variable defaults to `ON`.

- `query_cache_wlock_invalidate`

Normally, when one client acquires a `WRITE` lock on a `MyISAM` table, other clients are not blocked from issuing statements that read from the table if the query results are present in the query cache. Setting this variable to 1 causes acquisition of a `WRITE` lock for a table to invalidate any queries in the query cache that refer to the table. This forces other clients that attempt to access the table to wait while the lock is in effect.

- `query_prealloc_size`

The size of the persistent buffer used for statement parsing and execution. This buffer is not freed between statements. If you are running complex queries, a larger `query_prealloc_size` value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations.

- `range_alloc_block_size`

The size of blocks that are allocated when doing range optimization.

- `read_buffer_size`

Each thread that does a sequential scan allocates a buffer of this size (in bytes) for each table it scans. If you do many sequential scans, you might want to increase this value, which defaults to 131072.

`read_buffer_size` and `read_rnd_buffer_size` are not specific to any storage engine and apply in a general manner for optimization. See [Section 6.5.7, “How MySQL Uses Memory”](#), for example.

- `read_only`

When this variable is set to `ON`, the server allows no updates except from users that have the `SUPER` privilege or (on a slave server) from updates performed by slave threads. On a slave server, this can be useful to ensure that the slave accepts updates only from its master server and not from clients. This variable does not apply to `TEMPORARY` tables.

`read_only` exists only as a `GLOBAL` variable, so changes to its value require the `SUPER` privilege. Changes to `read_only` on a master server are not replicated to slave servers. The value can be set on a slave server independent of the setting on the master.

As of MySQL 5.1.15, the following conditions apply:

- If you attempt to enable `read_only` while you have any explicit locks (acquired with `LOCK TABLES` or have a pending transaction, an error will occur.
 - If other clients hold explicit table locks or have pending transactions, the attempt to enable `read_only` blocks until the locks are released and the transactions end. While the attempt to enable `read_only` is pending, requests by other clients for table locks or to begin transactions also block until `read_only` has been set.
 - `read_only` can be enabled while you hold a global read lock (acquired with `FLUSH TABLES WITH READ LOCK`) because that does not involve table locks.
- `read_rnd_buffer_size`

When reading rows in sorted order following a key-sorting operation, the rows are read through this buffer to avoid disk seeks. See [Section 6.2.11, “ORDER BY Optimization”](#). Setting the variable to a large value can improve `ORDER BY` performance by a lot.

However, this is a buffer allocated for each client, so you should not set the global variable to a large value. Instead, change the session variable only from within those clients that need to run large queries.

`read_buffer_size` and `read_rnd_buffer_size` are not specific to any storage engine and apply in a general manner for optimization. See [Section 6.5.7, “How MySQL Uses Memory”](#), for example.

- `relay_log_purge`

Disables or enables automatic purging of relay log files as soon as they are not needed any more. The default value is 1 (**ON**).

- `rpl_recovery_rank`

This variable is unused.

- `secure_auth`

If the MySQL server has been started with the `--secure-auth` option, it blocks connections from all accounts that have passwords stored in the old (pre-4.1) format. In that case, the value of this variable is **ON**, otherwise it is **OFF**.

You should enable this option if you want to prevent all use of passwords employing the old format (and hence insecure communication over the network).

Server startup fails with an error if this option is enabled and the privilege tables are in pre-4.1 format. See [Section B.1.2.3, “Client does not support authentication protocol”](#).

- `secure_file_priv`

By default, this variable is empty. If set to the name of a directory, it limits the effect of the `LOAD_FILE()` function and the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements to work only with files in that directory.

This variable was added in MySQL 5.1.17.

- `server_id`

The server ID. This value is set by the `--server-id` option. It is used for replication to enable master and slave servers to identify themselves uniquely.

- `shared_memory`

(Windows only.) Whether the server allows shared-memory connections.

- `shared_memory_base_name`

(Windows only.) The name of shared memory to use for shared-memory connections. This is useful when running multiple MySQL instances on a single physical machine. The default name is **MYSQL**. The name is case sensitive.

- `skip_external_locking`

This is **OFF** if `mysqld` uses external locking, **ON** if external locking is disabled.

- `skip_networking`

This is **ON** if the server allows only local (non-TCP/IP) connections. On Unix, local connections use a Unix socket file. On Windows, local connections use a named pipe or shared memory. On NetWare, only TCP/IP connections are supported, so do not set this variable to **ON**. This variable can be set to **ON** with the `--skip-networking` option.

- `skip_show_database`

This prevents people from using the `SHOW DATABASES` statement if they do not have the `SHOW DATABASES` privilege. This can improve security if you have concerns about users being able to see databases belonging to other users. Its effect depends on the `SHOW DATABASES` privilege: If the variable value is **ON**, the `SHOW DATABASES` statement is allowed only to users who have the `SHOW DATABASES` privilege, and the statement displays all database names. If the value is **OFF**, `SHOW DATABASES` is allowed to all users, but displays the names of only those databases for which the user has the `SHOW DATABASES` or other privilege.

- `slave_compressed_protocol`

Whether to use compression of the slave/master protocol if both the slave and the master support it.

- `slave_load_tmpdir`

The name of the directory where the slave creates temporary files for replicating `LOAD DATA INFILE` statements.

- `slave_net_timeout`

The number of seconds to wait for more data from a master/slave connection before aborting the read. This timeout applies only to TCP/IP connections, not to connections made via Unix socket files, named pipes, or shared memory.

- `slave_skip_errors`

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This variable tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the variable value.

- `slave_transaction_retries`

If a replication slave SQL thread fails to execute a transaction because of an InnoDB deadlock or exceeded InnoDB's `innodb_lock_wait_timeout` or NDBCluster's `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout`, it automatically retries `slave_transaction_retries` times before stopping with an error. The default value is 10.

- `slow_launch_time`

If creating a thread takes longer than this many seconds, the server increments the `Slow_launch_threads` status variable.

- `slow_query_log`

Whether the slow query log is enabled. The value can be 0 (or `OFF`) to disable the log or 1 (or `ON`) to enable the log. The default value depends on whether the `--log-slow-queries` option is given. The destination for log output is controlled by the `log_output` system variable; if that value is `NONE`, no log entries are written even if the log is enabled. The `slow_query_log` variable was added in MySQL 5.1.12.

- `slow_query_log_file`

The name of the slow query log file. The default value is `host_name-slow.log`, but the initial value can be changed with the `--log-slow-queries` option. This variable was added in MySQL 5.1.12.

- `socket`

On Unix platforms, this variable is the name of the socket file that is used for local client connections. The default is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

On Windows, this variable is the name of the named pipe that is used for local client connections. The default value is `MySQL` (not case sensitive).

- `sort_buffer_size`

Each thread that needs to do a sort allocates a buffer of this size. Increase this value for faster `ORDER BY` or `GROUP BY` operations. See [Section B.1.4.4, “Where MySQL Stores Temporary Files”](#).

- `sql_mode`

The current server SQL mode, which can be set dynamically. See [Section 5.2.6, “SQL Modes”](#).

- `sql_slave_skip_counter`

The number of events from the master that a slave server should skip. See [Section 12.6.2.6, “SET GLOBAL SQL_SLAVE_SKIP_COUNTER Syntax”](#).

- `ssl_ca`

The path to a file with a list of trusted SSL CAs. This variable was added in MySQL 5.1.11.

- `ssl_capath`

The path to a directory that contains trusted SSL CA certificates in PEM format. This variable was added in MySQL 5.1.11.

- `ssl_cert`

The name of the SSL certificate file to use for establishing a secure connection. This variable was added in MySQL 5.1.11.

- `ssl_cipher`

A list of allowable ciphers to use for SSL encryption. The cipher list has the same format as the `openssl ciphers` command. This variable was added in MySQL 5.1.11.

- `ssl_key`

The name of the SSL key file to use for establishing a secure connection. This variable was added in MySQL 5.1.11.

- `storage_engine`

The default storage engine (table type). To set the storage engine at server startup, use the `--default-storage-engine` option. See [Section 5.2.2, “Command Options”](#).

- `sync_binlog`

If the value of this variable is positive, the MySQL server synchronizes its binary log to disk (using `fdatasync()`) after every `sync_binlog` writes to the binary log. Note that there is one write to the binary log per statement if autocommit is enabled, and one write per transaction otherwise. The default value is 0, which does no synchronizing to disk. A value of 1 is the safest choice, because in the event of a crash you lose at most one statement or transaction from the binary log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

If the value of `sync_binlog` is 0 (the default), no extra flushing is done. The server relies on the operating system to flush the file contents occasionally as for any other file.

- `sync_frm`

If this variable is set to 1, when any non-temporary table is created its `.frm` file is synchronized to disk (using `fdatasync()`). This is slower but safer in case of a crash. The default is 1.

- `system_time_zone`

The server system time zone. When the server begins executing, it inherits a time zone setting from the machine defaults, possibly modified by the environment of the account used for running the server or the startup script. The value is used to set `sys-tem_time_zone`. Typically the time zone is specified by the `TZ` environment variable. It also can be specified using the `-timezone` option of the `mysqld_safe` script.

The `system_time_zone` variable differs from `time_zone`. Although they might have the same value, the latter variable is used to initialize the time zone for each client that connects. See [Section 5.10.8, “MySQL Server Time Zone Support”](#).

- `table_cache`

Name	<code>table_cache</code>	
Description	Deprecated; use <code>--table_open_cache</code> instead	
Option Sets Variable	Yes, <code>table_cache</code>	
Variable Name	<code>table_cache</code>	
Variable Scope	Server	
Dynamic Variable	yes	
Deprecated	5.1.3, by <code>table_open_cache</code>	
Value Set	Type	numeric
	Default	64

This is the old name of `table_open_cache` before MySQL 5.1.3. From 5.1.3 on, use `table_open_cache` instead.

- [table_definition_cache](#)

The number of table definitions that can be stored in the definition cache. If you use a large number of tables, you can create a large table definition cache to speed up opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the normal table cache. This variable was added in MySQL 5.1.3.

- [table_lock_wait_timeout](#)

Name	table_lock_wait_timeout	
Description	Timeout in seconds to wait for a table level lock before returning an error. Used only if the connection has active cursors	
Option Sets Variable	Yes, table_lock_wait_timeout	
Variable Name	table_lock_wait_timeout	
Variable Scope	Server	
Dynamic Variable	yes	
Value Set	Type	numeric
	Default	50

Specifies a wait timeout for table-level locks, in seconds. The default timeout is 50 seconds. The timeout is active only if the connection has open cursors. This variable can also be set globally at runtime (you need the [SUPER](#) privilege to do this).

- [table_open_cache](#)

The number of open tables for all threads. Increasing this value increases the number of file descriptors that `mysqld` requires. You can check whether you need to increase the table cache by checking the `Opened_tables` status variable. See [Section 5.2.5, “Status Variables”](#). If the value of `Opened_tables` is large and you don't do `FLUSH TABLES` often (which just forces all tables to be closed and reopened), then you should increase the value of the `table_open_cache` variable. For more information about the table cache, see [Section 6.4.8, “How MySQL Opens and Closes Tables”](#). Before MySQL 5.1.3, this variable is called `table_cache`.

- [table_type](#)

This variable is a synonym for `storage_engine`. In MySQL 5.1, `storage_engine` is the preferred name. In MySQL 5.2, `table_type` will be removed.

- [thread_cache_size](#)

How many threads the server should cache for reuse. When a client disconnects, the client's threads are put in the cache if there are fewer than `thread_cache_size` threads there. Requests for threads are satisfied by reusing threads taken from the cache if possible, and only when the cache is empty is a new thread created. This variable can be increased to improve performance if you have a lot of new connections. (Normally, this doesn't provide a notable performance improvement if you have a good thread implementation.) By examining the difference between the `Connections` and `Threads_created` status variables, you can see how efficient the thread cache is. For details, see [Section 5.2.5, “Status Variables”](#).

- [thread_concurrency](#)

On Solaris, `mysqld` calls `thr_setconcurrency()` with this value. This function enables applications to give the threads system a hint about the desired number of threads that should be run at the same time.

- [thread_handling](#)

The thread-handling model. The allowable values are `one-thread` (the server uses one thread) and `one-thread-per-connection` (the server uses multiple threads). `one-thread` is useful for debugging under Linux; see [MySQL Internals: Porting](#). This variable was added in MySQL 5.1.17

- [thread_stack](#)

The stack size for each thread. Many of the limits detected by the `crash-me` test are dependent on this value. The default is large enough for normal operation. See [Section 6.1.4, “The MySQL Benchmark Suite”](#). The default is 192KB.

- [time_format](#)

This variable is not implemented.

- `time_zone`

The current time zone. This variable is used to initialize the time zone for each client that connects. By default, the initial value of this is 'SYSTEM' (which means, “use the value of `system_time_zone`”). The value can be specified explicitly at server startup with the `--default-time-zone` option. See [Section 5.10.8, “MySQL Server Time Zone Support”](#).

- `timed_mutexes`

This variable controls whether InnoDB mutexes are timed. If this variable is set to 0 or OFF (the default), mutex timing is disabled. If the variable is set to 1 or ON, mutex timing is enabled. With timing enabled, the `os_wait_times` value in the output from `SHOW ENGINE INNODB MUTEX` indicates the amount of time (in ms) spent in operating system waits. Otherwise, the value is 0.

- `tmp_table_size`

The maximum size of internal in-memory temporary tables. (The actual limit is determined as the smaller of `max_heap_table_size` and `tmp_table_size`.) If an in-memory temporary table exceeds the limit, MySQL automatically converts it to an on-disk MyISAM table. Increase the value of `tmp_table_size` (and `max_heap_table_size` if necessary) if you do many advanced GROUP BY queries and you have lots of memory. This variable does not apply to user-created MEMORY tables.

- `tmpdir`

The directory used for temporary files and temporary tables. This variable can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (':') on Unix and semicolon characters(';') on Windows, NetWare, and OS/2.

The multiple-directory feature can be used to spread the load between several physical disks. If the MySQL server is acting as a replication slave, you should not set `tmpdir` to point to a directory on a memory-based filesystem or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or LOAD DATA INFILE operations. If files in the temporary file directory are lost when the server restarts, replication fails. However, if you are using MySQL 4.0.0 or later, you can set the slave's temporary directory using the `slave_load_tmpdir` variable. In that case, the slave won't use the general `tmpdir` value and you can set `tmpdir` to a non-permanent location.

- `transaction_alloc_block_size`

The amount in bytes by which to increase a per-transaction memory pool which needs memory. See the description of `transaction_prealloc_size`.

- `transaction_prealloc_size`

There is a per-transaction memory pool from which various transaction-related allocations take memory. The initial size of the pool in bytes is `transaction_prealloc_size`. For every allocation that cannot be satisfied from the pool because it has insufficient memory available, the pool is increased by `transaction_alloc_block_size` bytes. When the transaction ends, the pool is truncated to `transaction_prealloc_size` bytes.

By making `transaction_prealloc_size` sufficiently large to contain all statements within a single transaction, you can avoid many `malloc()` calls.

- `tx_isolation`

The default transaction isolation level. Defaults to REPEATABLE-READ.

This variable is set by the SET TRANSACTION ISOLATION LEVEL statement. See [Section 12.4.6, “SET TRANSACTION Syntax”](#). If you set `tx_isolation` directly to an isolation level name that contains a space, the name should be enclosed within quotes, with the space replaced by a dash. For example:

```
SET tx_isolation = 'READ-COMMITTED';
```

- `updatable_views_with_limit`

This variable controls whether updates to a view can be made when the view does not contain all columns of the primary key

defined in the underlying table, if the update statement contains a `LIMIT` clause. (Such updates often are generated by GUI tools.) An update is an `UPDATE` or `DELETE` statement. Primary key here means a `PRIMARY KEY`, or a `UNIQUE` index in which no column can contain `NULL`.

The variable can have two values:

- `1` or `YES`: Issue a warning only (not an error message). This is the default value.
- `0` or `NO`: Prohibit the update.
- `version`

The version number for the server.

- `version_comment`

The `configure` script has a `--with-comment` option that allows a comment to be specified when building MySQL. This variable contains the value of that comment.

- `version_compile_machine`

The type of machine or architecture on which MySQL was built.

- `version_compile_os`

The type of operating system on which MySQL was built.

- `wait_timeout`

The number of seconds the server waits for activity on a non-interactive connection before closing it. This timeout applies only to TCP/IP and Unix socket file connections, not to connections made via named pipes, or shared memory.

On thread startup, the session `wait_timeout` value is initialized from the global `wait_timeout` value or from the global `interactive_timeout` value, depending on the type of client (as defined by the `CLIENT_INTERACTIVE` connect option to `mysql_real_connect()`). See also `interactive_timeout`.

MySQL Enterprise

Expert use of server system variables is part of the service offered by the MySQL Enterprise Monitor. To subscribe see <http://www.mysql.com/products/enterprise/advisors.html>.

MySQL Enterprise

Expert use of server system variables is part of the service offered by the MySQL Enterprise Monitor. To subscribe see <http://www.mysql.com/products/enterprise/advisors.html>.

5.2.4. Using System Variables

The MySQL server maintains many system variables that indicate how it is configured. [Section 5.2.3, “System Variables”](#), describes the meaning of these variables. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

The server maintains two kinds of system variables. Global variables affect the overall operation of the server. Session variables affect its operation for individual client connections. A given system variable can have both a global and a session value. Global and session system variables are related as follows:

- When the server starts, it initializes all global variables to their default values. These defaults can be changed by options specified on the command line or in an option file. (See [Section 4.3, “Specifying Program Options”](#).)
- The server also maintains a set of session variables for each client that connects. The client's session variables are initialized at connect time using the current values of the corresponding global variables. For example, the client's SQL mode is controlled by the session `sql_mode` value, which is initialized when the client connects to the value of the global `sql_mode` value.

System variable values can be set globally at server startup by using options on the command line or in an option file. When you use a startup option to set a variable that takes a numeric value, the value can be given with a suffix of **K**, **M**, or **G** (either uppercase or lowercase) to indicate a multiplier of 1024, 1024² or 1024³; that is, units of kilobytes, megabytes, or gigabytes, respectively. Thus, the following command starts the server with a query cache size of 16 megabytes and a maximum packet size of one gigabyte:

```
mysqld --query_cache_size=16M --max_allowed_packet=1G
```

Within an option file, those variables are set like this:

```
[mysqld]
query_cache_size=16M
max_allowed_packet=1G
```

The lettercase of suffix letters does not matter; **16M** and **16m** are equivalent, as are **1G** and **1g**.

If you want to restrict the maximum value to which a system variable can be set at runtime with the **SET** statement, you can specify this maximum by using an option of the form **--maximum-var_name=value** at server startup. For example, to prevent the value of **query_cache_size** from being increased to more than 32MB at runtime, use the option **--maximum-query_cache_size=32M**.

Many system variables are dynamic and can be changed while the server runs by using the **SET** statement. For a list, see [Section 5.2.4.2, “Dynamic System Variables”](#). To change a system variable with **SET**, refer to it as **var_name**, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by **GLOBAL** or **@@global.**. The **SUPER** privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by **SESSION**, **@@session.**, or **@@**. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- **LOCAL** and **@@local.** are synonyms for **SESSION** and **@@session.**
- If no modifier is present, **SET** changes the session variable.

A **SET** statement can contain multiple variable assignments, separated by commas. If you set several system variables, the most recent **GLOBAL** or **SESSION** modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

When you assign a value to a system variable with **SET**, you cannot use suffix letters in the value (as can be done with startup options). However, the value can take the form of an expression:

```
SET sort_buffer_size = 10 * 1024 * 1024;
```

The **@@var_name** syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the **SET GLOBAL** statement).

To prevent incorrect usage, MySQL produces an error if you use **SET GLOBAL** with a variable that can only be used with **SET SESSION** or if you do not specify **GLOBAL** (or **@@global.**) when setting a global variable.

To set a **SESSION** variable to the **GLOBAL** value or a **GLOBAL** value to the compiled-in MySQL default value, use the **DEFAULT** keyword. For example, the following two statements are identical in setting the session value of **max_join_size** to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to **DEFAULT**. In such cases, use of **DEFAULT** results in an error.

You can refer to the values of specific global or session system variables in expressions by using one of the **@@**-modifiers. For example, you can retrieve values in a **SELECT** statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as **@@var_name** (that is, when you do not specify **@@global.** or **@@session.**), MySQL returns the session value if it exists and the global value otherwise. (This differs from **SET @@var_name = value**, which always refers to the session value.)

Note

Some system variables can be enabled with the **SET** statement by setting them to **ON** or **1**, or disabled by setting them to **OFF** or **0**. However, to set such a variable on the command line or in an option file, you must set it to **1** or **0**; setting it to **ON** or **OFF** will not work. For example, on the command line, **--delay_key_write=1** works but **-delay_key_write=ON** does not.

To display system variable names and values, use the **SHOW VARIABLES** statement:

```
mysql> SHOW VARIABLES;
```

Variable_name	Value
auto_increment_increment	1
auto_increment_offset	1
automatic_sp_privileges	ON
back_log	50
basedir	/home/mysql/
binlog_cache_size	32768
bulk_insert_buffer_size	8388608
character_set_client	latin1
character_set_connection	latin1
character_set_database	latin1
character_set_results	latin1
character_set_server	latin1
character_set_system	utf8
character_sets_dir	/home/mysql/share/mysql/charsets/
collation_connection	latin1_swedish_ci
collation_database	latin1_swedish_ci
collation_server	latin1_swedish_ci
...	
innodb_additional_mem_pool_size	1048576
innodb_autoextend_increment	8
innodb_buffer_pool_awe_mem_mb	0
innodb_buffer_pool_size	8388608
innodb_checksums	ON
innodb_commit_concurrency	0
innodb_concurrency_tickets	500
innodb_data_file_path	ibdata1:10M:autoextend
innodb_data_home_dir	
...	
version	5.1.6-alpha-log
version_comment	Source distribution
version_compile_machine	i686
version_compile_os	suse-linux
wait_timeout	28800

With a **LIKE** clause, the statement displays only those variables that match the pattern. To obtain a specific variable name, use a **LIKE** clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the **'%'** wildcard character in a **LIKE** clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because ‘_’ is a wildcard that matches any single character, you should escape it as ‘_’ to match it literally. In practice, this is rarely necessary.

For **SHOW VARIABLES**, if you specify neither **GLOBAL** nor **SESSION**, MySQL returns **SESSION** values.

The reason for requiring the **GLOBAL** keyword when setting **GLOBAL**-only variables but not when retrieving them is to prevent problems in the future. If we were to remove a **SESSION** variable that has the same name as a **GLOBAL** variable, a client with the **SUPER** privilege might accidentally change the **GLOBAL** variable rather than just the **SESSION** variable for its own connection. If we add a **SESSION** variable with the same name as a **GLOBAL** variable, a client that intends to change the **GLOBAL** variable might find only its own **SESSION** variable changed.

5.2.4.1. Structured System Variables

A structured variable differs from a regular system variable in two respects:

- Its value is a structure with components that specify server parameters considered to be closely related.
- There might be several instances of a given type of structured variable. Each one has a different name and refers to a different resource maintained by the server.

MySQL 5.1 supports one structured variable type, which specifies parameters governing the operation of key caches. A key cache structured variable has these components:

- `key_buffer_size`
- `key_cache_block_size`
- `key_cache_division_limit`
- `key_cache_age_threshold`

This section describes the syntax for referring to structured variables. Key cache variables are used for syntax examples, but specific details about how key caches operate are found elsewhere, in [Section 6.4.6, “The MyISAM Key Cache”](#).

To refer to a component of a structured variable instance, you can use a compound name in *instance_name.component_name* format. Examples:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

For each structured system variable, an instance with the name of `default` is always predefined. If you refer to a component of a structured variable without any instance name, the `default` instance is used. Thus, `default.key_buffer_size` and `key_buffer_size` both refer to the same system variable.

Structured variable instances and components follow these naming rules:

- For a given type of structured variable, each instance must have a name that is unique *within* variables of that type. However, instance names need not be unique *across* structured variable types. For example, each structured variable has an instance named `default`, so `default` is not unique across variable types.
- The names of the components of each structured variable type must be unique across all system variable names. If this were not true (that is, if two different types of structured variables could share component member names), it would not be clear which default structured variable to use for references to member names that are not qualified by an instance name.
- If a structured variable instance name is not legal as an unquoted identifier, refer to it as a quoted identifier using backticks. For example, `hot-cache` is not legal, but ``hot-cache`` is.
- `global`, `session`, and `local` are not legal instance names. This avoids a conflict with notation such as

`var_name` for referring to non-structured system variables.

Currently, the first two rules have no possibility of being violated because the only structured variable type is the one for key caches. These rules will assume greater significance if some other type of structured variable is created in the future.

With one exception, you can refer to structured variable components using compound names in any context where simple variable names can occur. For example, you can assign a value to a structured variable using a command-line option:

```
shell> mysql --hot_cache.key_buffer_size=64K
```

In an option file, use this syntax:

```
[mysql]  
hot_cache.key_buffer_size=64K
```

If you start the server with this option, it creates a key cache named `hot_cache` with a size of 64KB in addition to the default key cache that has a default size of 8MB.

Suppose that you start the server as follows:

```
shell> mysql --key_buffer_size=256K \  
--extra_cache.key_buffer_size=128K \  
--extra_cache.key_cache_block_size=2048
```

In this case, the server sets the size of the default key cache to 256KB. (You could also have written `-default.key_buffer_size=256K`.) In addition, the server creates a second key cache named `extra_cache` that has a size of 128KB, with the size of block buffers for caching table index blocks set to 2048 bytes.

The following example starts the server with three different key caches having sizes in a 3:1:1 ratio:

```
shell> mysql --key_buffer_size=6M \  
--hot_cache.key_buffer_size=2M \  
--cold_cache.key_buffer_size=2M
```

Structured variable values may be set and retrieved at runtime as well. For example, to set a key cache named `hot_cache` to a size of 10MB, use either of these statements:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;  
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

To retrieve the cache size, do this:

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

However, the following statement does not work. The variable is not interpreted as a compound name, but as a simple string for a `LIKE` pattern-matching operation:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

This is the exception to being able to use structured variable names anywhere a simple variable name may occur.

5.2.4.2. Dynamic System Variables

Many server system variables are dynamic and can be set at runtime using `SET GLOBAL` or `SET SESSION`. You can also obtain their values using `SELECT`. See [Section 5.2.4, “Using System Variables”](#).

The following table shows the full list of all dynamic system variables. The last column indicates for each variable whether `GLOBAL` or `SESSION` (or both) apply. The table also lists session options that can be set with the `SET` statement. [Section 12.5.3, “SET Syntax”](#), discusses these options.

Variables that have a type of “string” take a string value. Variables that have a type of “numeric” take a numeric value. Variables that have a type of “boolean” can be set to 0, 1, `ON` or `OFF`. (If you set them on the command line or in an option file, use the numeric values.) Variables that are marked as “enumeration” normally should be set to one of the available values for the variable, but can also be

set to the number that corresponds to the desired enumeration value. For enumerated system variables, the first enumeration value corresponds to 0. This differs from [ENUM](#) columns, for which the first enumeration value corresponds to 1.

Variable Name	Variable Type	Variable Scope
autocommit	boolean	SESSION
big_tables	boolean	GLOBAL SESSION
binlog_cache_size	numeric	GLOBAL SESSION
bulk_insert_buffer_size	numeric	GLOBAL SESSION
character_set_server	string	GLOBAL SESSION
character_set_client	string	GLOBAL SESSION
character_set_connection	string	GLOBAL SESSION
character_set_database	string	GLOBAL SESSION
character_set_results	string	GLOBAL SESSION
character_set_system	string	GLOBAL SESSION
collation_connection	string	GLOBAL SESSION
collation_database	string	GLOBAL SESSION
collation_server	string	GLOBAL SESSION
completion_type	numeric	GLOBAL SESSION
concurrent_insert	boolean	GLOBAL SESSION
connect_timeout	numeric	GLOBAL SESSION
date_format	string	GLOBAL SESSION
datetime_format	string	GLOBAL SESSION
default_week_format	numeric	GLOBAL SESSION
delay_key_write	enumeration	GLOBAL SESSION
delayed_insert_limit	numeric	GLOBAL SESSION
delayed_insert_timeout	numeric	GLOBAL SESSION
delayed_queue_size	numeric	GLOBAL SESSION
div_precision_increment	numeric	GLOBAL SESSION
engine_condition_pushdown	boolean	GLOBAL SESSION
expire_logs_days	numeric	GLOBAL
flush	boolean	GLOBAL SESSION
flush_time	numeric	GLOBAL SESSION
foreign_key_checks	boolean	SESSION
ft_boolean_syntax	string	GLOBAL SESSION
ft_max_word_len	numeric	GLOBAL SESSION
ft_min_word_len	numeric	GLOBAL
ft_query_expansion_limit	numeric	GLOBAL SESSION
ft_stopword_file	filename	GLOBAL SESSION
group_concat_max_len	numeric	GLOBAL SESSION
identity	numeric	SESSION
innodb_autoextend_increment	numeric	GLOBAL SESSION
innodb_autoinc_lock_mode	numeric	GLOBAL
innodb_checksums	boolean	GLOBAL SESSION
innodb_commit_concurrency	numeric	GLOBAL SESSION
innodb_concurrency_tickets	numeric	GLOBAL SESSION
innodb_flush_log_at_trx_commit	numeric	GLOBAL
innodb_max_dirty_pages_pct	numeric	GLOBAL SESSION

Variable Name	Variable Type	Variable Scope
<code>innodb_max_purge_lag</code>	numeric	GLOBAL SESSION
<code>innodb_support_xa</code>	boolean	GLOBAL SESSION
<code>innodb_sync_spin_loops</code>	numeric	GLOBAL SESSION
<code>innodb_table_locks</code>	boolean	GLOBAL SESSION
<code>innodb_thread_concurrency</code>	numeric	GLOBAL
<code>innodb_thread_sleep_delay</code>	numeric	GLOBAL SESSION
<code>insert_id</code>	numeric	SESSION
<code>interactive_timeout</code>	numeric	GLOBAL SESSION
<code>join_buffer_size</code>	numeric	GLOBAL SESSION
<code>keep_files_on_create</code>	boolean	GLOBAL
<code>key_buffer_size</code>	numeric	GLOBAL SESSION
<code>last_insert_id</code>	numeric	SESSION
<code>lc_time_names</code>	string	GLOBAL SESSION
<code>local_infile</code>	boolean	GLOBAL
<code>log_bin_trust_function_creators</code>	boolean	GLOBAL SESSION
<code>log_queries_not_using_indexes</code>	boolean	GLOBAL SESSION
<code>log_warnings</code>	numeric	GLOBAL SESSION
<code>log_output</code>	enumeration	GLOBAL
<code>long_query_time</code>	numeric	GLOBAL SESSION
<code>low_priority_updates</code>	numeric	GLOBAL SESSION
<code>max_allowed_packet</code>	numeric	GLOBAL SESSION
<code>max_binlog_cache_size</code>	numeric	GLOBAL
<code>max_binlog_size</code>	numeric	GLOBAL
<code>max_connect_errors</code>	numeric	GLOBAL
<code>max_connections</code>	numeric	GLOBAL
<code>max_delayed_threads</code>	numeric	GLOBAL
<code>max_error_count</code>	numeric	GLOBAL SESSION
<code>max_heap_table_size</code>	numeric	GLOBAL SESSION
<code>max_insert_delayed_threads</code>	numeric	GLOBAL
<code>max_join_size</code>	numeric	GLOBAL SESSION
<code>max_length_for_sort_data</code>	numeric	GLOBAL SESSION
<code>max_prepared_stmt_count</code>	numeric	GLOBAL
<code>max_relay_log_size</code>	numeric	GLOBAL
<code>max_seeks_for_key</code>	numeric	GLOBAL SESSION
<code>max_sort_length</code>	numeric	GLOBAL SESSION
<code>max_sp_recursion_depth</code>	numeric	GLOBAL
<code>max_tmp_tables</code>	numeric	GLOBAL SESSION
<code>max_user_connections</code>	numeric	GLOBAL SESSION
<code>max_write_lock_count</code>	numeric	GLOBAL
<code>multi_range_count</code>	numeric	GLOBAL SESSION
<code>multi_read_range</code>	numeric	GLOBAL SESSION
<code>myisam_block_size</code>	numeric	GLOBAL SESSION
<code>myisam_data_pointer_size</code>	numeric	GLOBAL
<code>myisam_max_sort_file_size</code>	numeric	GLOBAL SESSION

Variable Name	Variable Type	Variable Scope
myisam_repair_threads	numeric	GLOBAL SESSION
myisam_sort_buffer_size	numeric	GLOBAL SESSION
myisam_stats_method	enumeration	GLOBAL SESSION
ndb_autoincrement_prefetch_sz	numeric	GLOBAL SESSION
ndb_cache_check_time	numeric	GLOBAL SESSION
ndb_extra_logging	numeric	GLOBAL
ndb_force_send	boolean	GLOBAL SESSION
ndb_use_exact_count	boolean	GLOBAL SESSION
ndbcluster	boolean	GLOBAL SESSION
net_buffer_length	numeric	GLOBAL SESSION
net_read_timeout	numeric	GLOBAL SESSION
net_retry_count	numeric	GLOBAL SESSION
net_write_timeout	numeric	GLOBAL SESSION
old_passwords	boolean	GLOBAL SESSION
optimizer_prune_level	boolean	GLOBAL SESSION
optimizer_search_depth	numeric	GLOBAL SESSION
plugin_innodb_autoextend_increment	numeric	GLOBAL SESSION
plugin_innodb_checksums	boolean	GLOBAL SESSION
plugin_innodb_commit_concurrency	numeric	GLOBAL SESSION
plugin_innodb_concurrency_tickets	numeric	GLOBAL SESSION
plugin_innodb_flush_log_at_trx_commit	numeric	GLOBAL
plugin_innodb_max_dirty_pages_pct	numeric	GLOBAL SESSION
plugin_innodb_max_purge_lag	numeric	GLOBAL SESSION
plugin_innodb_support_xa	boolean	GLOBAL SESSION
innodb_sync_spin_loops	numeric	GLOBAL SESSION
plugin_innodb_table_locks	boolean	GLOBAL SESSION
plugin_innodb_thread_concurrency	numeric	GLOBAL SESSION
plugin_innodb_thread_sleep_delay	numeric	GLOBAL SESSION
preload_buffer_size	numeric	GLOBAL SESSION
query_alloc_block_size	numeric	GLOBAL SESSION
query_cache_limit	numeric	GLOBAL SESSION
query_cache_min_res_unit	numeric	GLOBAL SESSION
query_cache_size	numeric	GLOBAL SESSION
query_cache_type	enumeration	GLOBAL SESSION
query_cache_wlock_invalidate	boolean	GLOBAL SESSION
query_prealloc_size	numeric	GLOBAL SESSION
range_alloc_block_size	numeric	GLOBAL SESSION
read_only	numeric	GLOBAL SESSION
read_rnd_buffer_size	numeric	GLOBAL SESSION
rpl_recovery_rank	numeric	GLOBAL SESSION
secure_auth	boolean	GLOBAL SESSION
server_id	numeric	GLOBAL SESSION
slave_allow_batching	boolean	GLOBAL
slave_net_timeout	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
slave_transaction_retries	numeric	GLOBAL SESSION
slow_launch_time	numeric	GLOBAL SESSION
sort_buffer_size	numeric	GLOBAL SESSION
sql_mode	enumeration	GLOBAL SESSION
sql_big_selects	boolean	GLOBAL SESSION
sql_big_tables	boolean	SESSION
sql_buffer_result	boolean	SESSION
sql_log_bin	boolean	SESSION
sql_log_off	boolean	SESSION
sql_log_update	boolean	SESSION
sql_low_priority_updates	boolean	GLOBAL SESSION
sql_max_join_size	numeric	GLOBAL SESSION
sql_notes	boolean	GLOBAL SESSION
sql_quote_show_create	boolean	SESSION
sql_safe_updates	boolean	SESSION
sql_select_limit	numeric	SESSION
sql_slave_skip_counter	numeric	GLOBAL
sql_warnings	boolean	GLOBAL SESSION
sync_binlog	numeric	GLOBAL SESSION
sync_frm	boolean	GLOBAL SESSION
table_cache	numeric	GLOBAL SESSION
table_lock_wait_timeout	numeric	GLOBAL SESSION
table_open_cache	numeric	GLOBAL
table_type	enumeration	GLOBAL SESSION
thread_cache_size	numeric	GLOBAL SESSION
time_zone	string	GLOBAL SESSION
timestamp	string	SESSION
tmp_table_size	numeric	GLOBAL SESSION
transaction_alloc_block_size	numeric	GLOBAL SESSION
transaction_prealloc_size	numeric	GLOBAL SESSION
tx_isolation	enumeration	GLOBAL SESSION
unique_checks	boolean	SESSION
wait_timeout	numeric	GLOBAL SESSION

MySQL Enterprise

Improper configuration of system variables can adversely affect performance and security. The MySQL Enterprise Monitor continually monitors system variables and provides expert advice about appropriate settings. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

5.2.5. Status Variables

The server maintains many status variables that provide information about its operation. You can view these variables and their values by using the `SHOW [GLOBAL] STATUS` statement. The optional `GLOBAL` keyword aggregates the values over all connections.

```
mysql> SHOW GLOBAL STATUS;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Aborted_clients | 0 |
+-----+-----+
```

Aborted_connects	0
Bytes_received	155372598
Bytes_sent	1176560426
Connections	30023
Created_tmp_disk_tables	0
Created_tmp_files	3
Created_tmp_tables	2
Threads_created	217
Threads_running	88
Uptime	1389872

The following table lists all available server status variables:

Variable Name	Variable Type	Variable Scope
Aborted_clients	numeric	GLOBAL
Aborted_connects	numeric	GLOBAL
Binlog_cache_disk_use	numeric	GLOBAL
Binlog_cache_use	numeric	GLOBAL
Bytes_received	numeric	GLOBAL SESSION
Bytes_sent	numeric	GLOBAL SESSION
Com_admin_commands	numeric	GLOBAL SESSION
Com_alter_db	numeric	GLOBAL SESSION
Com_alter_event	numeric	GLOBAL SESSION
Com_alter_table	numeric	GLOBAL SESSION
Com_analyze	numeric	GLOBAL SESSION
Com_backup_table	numeric	GLOBAL SESSION
Com_begin	numeric	GLOBAL SESSION
Com_call_procedure	numeric	GLOBAL SESSION
Com_change_db	numeric	GLOBAL SESSION
Com_change_master	numeric	GLOBAL SESSION
Com_check	numeric	GLOBAL SESSION
Com_checksum	numeric	GLOBAL SESSION
Com_commit	numeric	GLOBAL SESSION
Com_create_db	numeric	GLOBAL SESSION
Com_create_event	numeric	GLOBAL SESSION
Com_create_function	numeric	GLOBAL SESSION
Com_create_index	numeric	GLOBAL SESSION
Com_create_table	numeric	GLOBAL SESSION
Com_create_user	numeric	GLOBAL SESSION
Com_dealloc_sql	numeric	GLOBAL SESSION
Com_delete	numeric	GLOBAL SESSION
Com_delete_multi	numeric	GLOBAL SESSION
Com_do	numeric	GLOBAL SESSION
Com_drop_db	numeric	GLOBAL SESSION
Com_drop_event	numeric	GLOBAL SESSION
Com_drop_function	numeric	GLOBAL SESSION
Com_drop_index	numeric	GLOBAL SESSION
Com_drop_table	numeric	GLOBAL SESSION
Com_drop_user	numeric	GLOBAL SESSION

Variable Name	Variable Type	Variable Scope
Com_execute_sql	numeric	GLOBAL SESSION
Com_flush	numeric	GLOBAL SESSION
Com_grant	numeric	GLOBAL SESSION
Com_ha_close	numeric	GLOBAL SESSION
Com_ha_open	numeric	GLOBAL SESSION
Com_ha_read	numeric	GLOBAL SESSION
Com_help	numeric	GLOBAL SESSION
Com_insert	numeric	GLOBAL SESSION
Com_insert_select	numeric	GLOBAL SESSION
Com_kill	numeric	GLOBAL SESSION
Com_load	numeric	GLOBAL SESSION
Com_lock_tables	numeric	GLOBAL SESSION
Com_optimize	numeric	GLOBAL SESSION
Com_preload_keys	numeric	GLOBAL SESSION
Com_prepare_sql	numeric	GLOBAL SESSION
Com_purge	numeric	GLOBAL SESSION
Com_purge_before_date	numeric	GLOBAL SESSION
Com_rename_table	numeric	GLOBAL SESSION
Com_repair	numeric	GLOBAL SESSION
Com_replace	numeric	GLOBAL SESSION
Com_replace_select	numeric	GLOBAL SESSION
Com_reset	numeric	GLOBAL SESSION
Com_restore_table	numeric	GLOBAL SESSION
Com_revoke	numeric	GLOBAL SESSION
Com_revoke_all	numeric	GLOBAL SESSION
Com_rollback	numeric	GLOBAL SESSION
Com_savepoint	numeric	GLOBAL SESSION
Com_select	numeric	GLOBAL SESSION
Com_set_option	numeric	GLOBAL SESSION
Com_show_binlog_events	numeric	GLOBAL SESSION
Com_show_binlogs	numeric	GLOBAL SESSION
Com_show_charsets	numeric	GLOBAL SESSION
Com_show_collations	numeric	GLOBAL SESSION
Com_show_column_types	numeric	GLOBAL SESSION
Com_show_create_db	numeric	GLOBAL SESSION
Com_show_create_event	numeric	GLOBAL SESSION
Com_show_create_table	numeric	GLOBAL SESSION
Com_show_databases	numeric	GLOBAL SESSION
Com_show_engine_logs	numeric	GLOBAL SESSION
Com_show_engine_mutex	numeric	GLOBAL SESSION
Com_show_engine_status	numeric	GLOBAL SESSION
Com_show_errors	numeric	GLOBAL SESSION
Com_show_events	numeric	GLOBAL SESSION
Com_show_fields	numeric	GLOBAL SESSION

Variable Name	Variable Type	Variable Scope
Com_show_grants	numeric	GLOBAL SESSION
Com_show_keys	numeric	GLOBAL SESSION
Com_show_master_status	numeric	GLOBAL SESSION
Com_show_new_master	numeric	GLOBAL SESSION
Com_show_open_tables	numeric	GLOBAL SESSION
Com_show_plugins	numeric	GLOBAL SESSION
Com_show_privileges	numeric	GLOBAL SESSION
Com_show_processlist	numeric	GLOBAL SESSION
Com_show_slave_hosts	numeric	GLOBAL SESSION
Com_show_slave_status	numeric	GLOBAL SESSION
Com_show_status	numeric	GLOBAL SESSION
Com_show_storage_engines	numeric	GLOBAL SESSION
Com_show_tables	numeric	GLOBAL SESSION
Com_show_triggers	numeric	GLOBAL SESSION
Com_show_variables	numeric	GLOBAL SESSION
Com_show_warnings	numeric	GLOBAL SESSION
Com_slave_start	numeric	GLOBAL SESSION
Com_slave_stop	numeric	GLOBAL SESSION
Com_stmt_close	numeric	GLOBAL SESSION
Com_stmt_execute	numeric	GLOBAL SESSION
Com_stmt_fetch	numeric	GLOBAL SESSION
Com_stmt_prepare	numeric	GLOBAL SESSION
Com_stmt_reset	numeric	GLOBAL SESSION
Com_stmt_send_long_data	numeric	GLOBAL SESSION
Com_truncate	numeric	GLOBAL SESSION
Com_unlock_tables	numeric	GLOBAL SESSION
Com_update	numeric	GLOBAL SESSION
Com_update_multi	numeric	GLOBAL SESSION
Com_xa_commit	numeric	GLOBAL SESSION
Com_xa_end	numeric	GLOBAL SESSION
Com_xa_prepare	numeric	GLOBAL SESSION
Com_xa_recover	numeric	GLOBAL SESSION
Com_xa_rollback	numeric	GLOBAL SESSION
Com_xa_start	numeric	GLOBAL SESSION
Compression	numeric	SESSION
Connections	numeric	GLOBAL
Created_tmp_disk_tables	numeric	GLOBAL SESSION
Created_tmp_files	numeric	GLOBAL
Created_tmp_tables	numeric	GLOBAL SESSION
Delayed_errors	numeric	GLOBAL
Delayed_insert_threads	numeric	GLOBAL
Delayed_writes	numeric	GLOBAL
Flush_commands	numeric	GLOBAL
Handler_commit	numeric	GLOBAL SESSION

Variable Name	Variable Type	Variable Scope
Handler_delete	numeric	GLOBAL SESSION
Handler_discover	numeric	GLOBAL SESSION
Handler_prepare	numeric	GLOBAL SESSION
Handler_read_first	numeric	GLOBAL SESSION
Handler_read_key	numeric	GLOBAL SESSION
Handler_read_next	numeric	GLOBAL SESSION
Handler_read_prev	numeric	GLOBAL SESSION
Handler_read_rnd	numeric	GLOBAL SESSION
Handler_read_rnd_next	numeric	GLOBAL SESSION
Handler_rollback	numeric	GLOBAL SESSION
Handler_savepoint	numeric	GLOBAL SESSION
Handler_savepoint_rollback	numeric	GLOBAL SESSION
Handler_update	numeric	GLOBAL SESSION
Handler_write	numeric	GLOBAL SESSION
Innodb_buffer_pool_pages_data	numeric	GLOBAL SESSION
Innodb_buffer_pool_pages_dirty	numeric	GLOBAL SESSION
Innodb_buffer_pool_pages_flushed	numeric	GLOBAL SESSION
Innodb_buffer_pool_pages_free	numeric	GLOBAL SESSION
Innodb_buffer_pool_pages_latched	numeric	GLOBAL SESSION
Innodb_buffer_pool_pages_misc	numeric	GLOBAL SESSION
Innodb_buffer_pool_pages_total	numeric	GLOBAL SESSION
Innodb_buffer_pool_read_ahead_rnd	numeric	GLOBAL SESSION
Innodb_buffer_pool_read_ahead_seq	numeric	GLOBAL SESSION
Innodb_buffer_pool_read_requests	numeric	GLOBAL SESSION
Innodb_buffer_pool_reads	numeric	GLOBAL SESSION
Innodb_buffer_pool_wait_free	numeric	GLOBAL SESSION
Innodb_buffer_pool_write_requests	numeric	GLOBAL SESSION
Innodb_data_fsyncs	numeric	GLOBAL SESSION
Innodb_data_pending_fsyncs	numeric	GLOBAL SESSION
Innodb_data_pending_reads	numeric	GLOBAL SESSION
Innodb_data_pending_writes	numeric	GLOBAL SESSION
Innodb_data_read	numeric	GLOBAL SESSION
Innodb_data_reads	numeric	GLOBAL SESSION
Innodb_data_writes	numeric	GLOBAL SESSION
Innodb_data_written	numeric	GLOBAL SESSION
Innodb_dblwr_pages_written	numeric	GLOBAL SESSION
Innodb_dblwr_writes	numeric	GLOBAL SESSION
Innodb_log_waits	numeric	GLOBAL SESSION
Innodb_log_write_requests	numeric	GLOBAL SESSION
Innodb_log_writes	numeric	GLOBAL SESSION
Innodb_os_log_fsyncs	numeric	GLOBAL SESSION
Innodb_os_log_pending_fsyncs	numeric	GLOBAL SESSION
Innodb_os_log_pending_writes	numeric	GLOBAL SESSION
Innodb_os_log_written	numeric	GLOBAL SESSION

Variable Name	Variable Type	Variable Scope
Innodb_page_size	numeric	GLOBAL SESSION
Innodb_pages_created	numeric	GLOBAL SESSION
Innodb_pages_read	numeric	GLOBAL SESSION
Innodb_pages_written	numeric	GLOBAL SESSION
Innodb_row_lock_current_waits	numeric	GLOBAL SESSION
Innodb_row_lock_time	numeric	GLOBAL SESSION
Innodb_row_lock_time_avg	numeric	GLOBAL SESSION
Innodb_row_lock_time_max	numeric	GLOBAL SESSION
Innodb_row_lock_waits	numeric	GLOBAL SESSION
Innodb_rows_deleted	numeric	GLOBAL SESSION
Innodb_rows_inserted	numeric	GLOBAL SESSION
Innodb_rows_read	numeric	GLOBAL SESSION
Innodb_rows_updated	numeric	GLOBAL SESSION
Key_blocks_not_flushed	numeric	GLOBAL SESSION
Key_blocks_unused	numeric	GLOBAL SESSION
Key_blocks_used	numeric	GLOBAL SESSION
Key_read_requests	numeric	GLOBAL SESSION
Key_reads	numeric	GLOBAL SESSION
Key_write_requests	numeric	GLOBAL SESSION
Key_writes	numeric	GLOBAL SESSION
Last_query_cost	numeric	GLOBAL SESSION
Max_used_connections	numeric	GLOBAL
Ndb_cluster_node_id	numeric	GLOBAL SESSION
Ndb_config_from_host	numeric	GLOBAL SESSION
Ndb_config_from_port	numeric	GLOBAL SESSION
Ndb_number_of_data_nodes	numeric	GLOBAL
Not_flushed_delayed_rows	numeric	GLOBAL
Open_files	numeric	GLOBAL
Open_streams	numeric	GLOBAL
Open_table_definitions	numeric	GLOBAL
Open_tables	numeric	GLOBAL
Opened_tables	numeric	GLOBAL SESSION
prepared_stmt_count	numeric	GLOBAL SESSION
Qcache_free_blocks	numeric	GLOBAL
Qcache_free_memory	numeric	GLOBAL
Qcache_hits	numeric	GLOBAL
Qcache_inserts	numeric	GLOBAL
Qcache_lowmem_prunes	numeric	GLOBAL
Qcache_not_cached	numeric	GLOBAL
Qcache_queries_in_cache	numeric	GLOBAL
Qcache_total_blocks	numeric	GLOBAL
Questions	numeric	GLOBAL
Relay_Log_Space	numeric	GLOBAL SESSION
Rpl_status		GLOBAL

Variable Name	Variable Type	Variable Scope
Select_full_join	numeric	GLOBAL SESSION
Select_full_range_join	numeric	GLOBAL SESSION
Select_range	numeric	GLOBAL SESSION
Select_range_check	numeric	GLOBAL SESSION
Select_scan	numeric	GLOBAL SESSION
Slave_open_temp_tables	numeric	GLOBAL
Slave_retried_transactions	numeric	GLOBAL
Slave_running	boolean	GLOBAL
Slow_launch_threads	numeric	GLOBAL
Slow_queries	numeric	GLOBAL SESSION
Sort_merge_passes	numeric	GLOBAL SESSION
Sort_range	numeric	GLOBAL SESSION
Sort_rows	numeric	GLOBAL SESSION
Sort_scan	numeric	GLOBAL SESSION
Table_locks_immediate	numeric	GLOBAL
Table_locks_waited	numeric	GLOBAL
Tc_log_max_pages_used	numeric	GLOBAL
Tc_log_page_size	numeric	GLOBAL
Tc_log_page_waits	numeric	GLOBAL
Threads_cached	numeric	GLOBAL
Threads_connected	numeric	GLOBAL
Threads_created	numeric	GLOBAL
Threads_running	numeric	GLOBAL
Uptime	numeric	GLOBAL

Many status variables are reset to 0 by the `FLUSH STATUS` statement.

MySQL Enterprise

For expert advice on using status variables, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

The status variables have the following meanings. Variables with no version indicated were already present prior to MySQL 5.1. For information regarding their implementation history, see *MySQL 5.0 Reference Manual*.

- Aborted_clients

The number of connections that were aborted because the client died without closing the connection properly. See [Section B.1.2.10, “Communication Errors and Aborted Connections”](#).

- Aborted_connects

The number of failed attempts to connect to the MySQL server. See [Section B.1.2.10, “Communication Errors and Aborted Connections”](#).

- Binlog_cache_disk_use

The number of transactions that used the temporary binary log cache but that exceeded the value of `binlog_cache_size` and used a temporary file to store statements from the transaction.

- Binlog_cache_use

The number of transactions that used the temporary binary log cache.

- **Bytes_received**

The number of bytes received from all clients.

- **Bytes_sent**

The number of bytes sent to all clients.

- **Com_xxx**

The **Com_xxx** statement counter variables indicate the number of times each **xxx** statement has been executed. There is one status variable for each type of statement. For example, **Com_delete** and **Com_insert** count **DELETE** and **INSERT** statements, respectively. However, if a query result is returned from query cache, the server increments the **Qcache_hits** status variable, not **Com_select**. See [Section 6.5.4.4, “Query Cache Status and Maintenance”](#).

All of the **Com_STATEMENT_xxx** variables are increased even if a prepared statement argument is unknown or an error occurred during execution. In other words, their values correspond to the number of requests issued, not to the number of requests successfully completed.

The **Com_STATEMENT_xxx** status variables are as follows:

- **Com_STATEMENT_prepare**
- **Com_STATEMENT_execute**
- **Com_STATEMENT_fetch**
- **Com_STATEMENT_send_long_data**
- **Com_STATEMENT_reset**
- **Com_STATEMENT_close**

Those variables stand for prepared statement commands. Their names refer to the **COM_xxx** command set used in the network layer. In other words, their values increase whenever prepared statement API calls such as **mysql_STATEMENT_prepare()**, **mysql_STATEMENT_execute()**, and so forth are executed. However, **Com_STATEMENT_prepare**, **Com_STATEMENT_execute** and **Com_STATEMENT_close** also increase for **PREPARE**, **EXECUTE**, or **DEALLOCATE PREPARE**, respectively. Additionally, the values of the older (available since MySQL 4.1.3) statement counter variables **Com_prepare_sql**, **Com_execute_sql**, and **Com_dealloc_sql** increase for the **PREPARE**, **EXECUTE**, and **DEALLOCATE PREPARE** statements. **Com_STATEMENT_fetch** stands for the total number of network round-trips issued when fetching from cursors.

- **Compression**

Whether the client connection uses compression in the client/server protocol. Added in MySQL 5.1.2.

- **Connections**

The number of connection attempts (successful or not) to the MySQL server.

- **Created_tmp_disk_tables**

The number of temporary tables on disk created automatically by the server while executing statements.

- **Created_tmp_files**

How many temporary files **mysqld** has created.

- **Created_tmp_tables**

The number of in-memory temporary tables created automatically by the server while executing statements. If **Created_tmp_disk_tables** is large, you may want to increase the **tmp_table_size** value to cause temporary tables to be memory-based instead of disk-based.

- `Delayed_errors`

The number of rows written with `INSERT DELAYED` for which some error occurred (probably `duplicate key`).

- `Delayed_insert_threads`

The number of `INSERT DELAYED` handler threads in use.

- `Delayed_writes`

The number of `INSERT DELAYED` rows written.

- `Flush_commands`

The number of executed `FLUSH` statements.

- `Handler_commit`

The number of internal `COMMIT` statements.

- `Handler_delete`

The number of times that rows have been deleted from tables.

- `Handler_discover`

The MySQL server can ask the `NDB Cluster` storage engine if it knows about a table with a given name. This is called discovery. `Handler_discover` indicates the number of times that tables have been discovered via this mechanism.

- `Handler_prepare`

A counter for the prepare phase of two-phase commit operations.

- `Handler_read_first`

The number of times the first entry was read from an index. If this value is high, it suggests that the server is doing a lot of full index scans; for example, `SELECT col1 FROM foo`, assuming that `col1` is indexed.

- `Handler_read_key`

The number of requests to read a row based on a key. If this value is high, it is a good indication that your tables are properly indexed for your queries.

- `Handler_read_next`

The number of requests to read the next row in key order. This value is incremented if you are querying an index column with a range constraint or if you are doing an index scan.

- `Handler_read_prev`

The number of requests to read the previous row in key order. This read method is mainly used to optimize `ORDER BY ... DESC`.

- `Handler_read_rnd`

The number of requests to read a row based on a fixed position. This value is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan entire tables or you have joins that don't use keys properly.

- `Handler_read_rnd_next`

The number of requests to read the next row in the data file. This value is high if you are doing a lot of table scans. Generally this suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.

- `Handler_rollback`

The number of requests for a storage engine to perform a rollback operation.

- `Handler_savepoint`
The number of requests for a storage engine to place a savepoint.
- `Handler_savepoint_rollback`
The number of requests for a storage engine to roll back to a savepoint.
- `Handler_update`
The number of requests to update a row in a table.
- `Handler_write`
The number of requests to insert a row in a table.
- `Innodb_buffer_pool_pages_data`
The number of pages containing data (dirty or clean).
- `Innodb_buffer_pool_pages_dirty`
The number of pages currently dirty.
- `Innodb_buffer_pool_pages_flushed`
The number of buffer pool page-flush requests.
- `Innodb_buffer_pool_pages_free`
The number of free pages.
- `Innodb_buffer_pool_pages_latched`
The number of latched pages in `InnoDB` buffer pool. These are pages currently being read or written or that cannot be flushed or removed for some other reason.
- `Innodb_buffer_pool_pages_misc`
The number of pages that are busy because they have been allocated for administrative overhead such as row locks or the adaptive hash index. This value can also be calculated as `Innodb_buffer_pool_pages_total - Innodb_buffer_pool_pages_free - Innodb_buffer_pool_pages_data`.
- `Innodb_buffer_pool_pages_total`
The total size of the buffer pool, in pages.
- `Innodb_buffer_pool_read_ahead_rnd`
The number of “random” read-aheads initiated by `InnoDB`. This happens when a query scans a large portion of a table but in random order.
- `Innodb_buffer_pool_read_ahead_seq`
The number of sequential read-aheads initiated by `InnoDB`. This happens when `InnoDB` does a sequential full table scan.
- `Innodb_buffer_pool_read_requests`
The number of logical read requests `InnoDB` has done.
- `Innodb_buffer_pool_reads`
The number of logical reads that `InnoDB` could not satisfy from the buffer pool and had to do a single-page read.
- `Innodb_buffer_pool_wait_free`
Normally, writes to the `InnoDB` buffer pool happen in the background. However, if it is necessary to read or create a page and no

clean pages are available, it is also necessary to wait for pages to be flushed first. This counter counts instances of these waits. If the buffer pool size has been set properly, this value should be small.

- [Innodb_buffer_pool_write_requests](#)

The number writes done to the [InnoDB](#) buffer pool.

- [Innodb_data_fsyncs](#)

The number of `fsync()` operations so far.

- [Innodb_data_pending_fsyncs](#)

The current number of pending `fsync()` operations.

- [Innodb_data_pending_reads](#)

The current number of pending reads.

- [Innodb_data_pending_writes](#)

The current number of pending writes.

- [Innodb_data_read](#)

The amount of data read so far, in bytes.

- [Innodb_data_reads](#)

The total number of data reads.

- [Innodb_data_writes](#)

The total number of data writes.

- [Innodb_data_written](#)

The amount of data written so far, in bytes.

- [Innodb_dblwr_writes](#), [Innodb_dblwr_pages_written](#)

The number of doublewrite operations that have been performed and the number of pages that have been written for this purpose. See [Section 13.5.14.1, “InnoDB Disk I/O”](#).

- [Innodb_log_waits](#)

The number of times that the log buffer was too small and a wait was required for it to be flushed before continuing.

- [Innodb_log_write_requests](#)

The number of log write requests.

- [Innodb_log_writes](#)

The number of physical writes to the log file.

- [Innodb_os_log_fsyncs](#)

The number of `fsync()` writes done to the log file.

- [Innodb_os_log_pending_fsyncs](#)

The number of pending log file `fsync()` operations.

- [Innodb_os_log_pending_writes](#)

The number of pending log file writes.

- [Innodb_os_log_written](#)
The number of bytes written to the log file.
- [Innodb_page_size](#)
The compiled-in [InnoDB](#) page size (default 16KB). Many values are counted in pages; the page size allows them to be easily converted to bytes.
- [Innodb_pages_created](#)
The number of pages created.
- [Innodb_pages_read](#)
The number of pages read.
- [Innodb_pages_written](#)
The number of pages written.
- [Innodb_row_lock_current_waits](#)
The number of row locks currently being waited for.
- [Innodb_row_lock_time](#)
The total time spent in acquiring row locks, in milliseconds.
- [Innodb_row_lock_time_avg](#)
The average time to acquire a row lock, in milliseconds.
- [Innodb_row_lock_time_max](#)
The maximum time to acquire a row lock, in milliseconds.
- [Innodb_row_lock_waits](#)
The number of times a row lock had to be waited for.
- [Innodb_rows_deleted](#)
The number of rows deleted from [InnoDB](#) tables.
- [Innodb_rows_inserted](#)
The number of rows inserted into [InnoDB](#) tables.
- [Innodb_rows_read](#)
The number of rows read from [InnoDB](#) tables.
- [Innodb_rows_updated](#)
The number of rows updated in [InnoDB](#) tables.
- [Key_blocks_not_flushed](#)
The number of key blocks in the key cache that have changed but have not yet been flushed to disk.
- [Key_blocks_unused](#)
The number of unused blocks in the key cache. You can use this value to determine how much of the key cache is in use; see the discussion of [key_buffer_size](#) in [Section 5.2.3, “System Variables”](#).
- [Key_blocks_used](#)

The number of used blocks in the key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.

- **Key_read_requests**

The number of requests to read a key block from the cache.

- **Key_reads**

The number of physical reads of a key block from disk. If **Key_reads** is large, then your **key_buffer_size** value is probably too small. The cache miss rate can be calculated as **Key_reads/Key_read_requests**.

- **Key_write_requests**

The number of requests to write a key block to the cache.

- **Key_writes**

The number of physical writes of a key block to disk.

- **Last_query_cost**

The total cost of the last compiled query as computed by the query optimizer. This is useful for comparing the cost of different query plans for the same query. The default value of 0 means that no query has been compiled yet. The default value is 0. **Last_query_cost** has session scope.

- **Max_used_connections**

The maximum number of connections that have been in use simultaneously since the server started.

- **Ndb_cluster_node_id**

If the server is acting as a MySQL Cluster node, then the value of this variable is its node ID in the cluster.

If the server is not part of a MySQL Cluster, then the value of this variable is 0.

- **Ndb_config_from_host**

If the server is part of a MySQL Cluster, the value of this variable is the hostname or IP address of the Cluster management server from which it gets its configuration data.

If the server is not part of a MySQL Cluster, then the value of this variable is an empty string.

Prior to MySQL 5.1.12, this variable was named **Ndb_connected_host**.

- **Ndb_config_from_port**

If the server is part of a MySQL Cluster, the value of this variable is the number of the port through which it is connected to the Cluster management server from which it gets its configuration data.

If the server is not part of a MySQL Cluster, then the value of this variable is 0.

Prior to MySQL 5.1.12, this variable was named **Ndb_connected_port**.

- **Ndb_number_of_data_nodes**

If the server is part of a MySQL Cluster, the value of this variable is the number of data nodes in the cluster.

If the server is not part of a MySQL Cluster, then the value of this variable is 0.

Prior to MySQL 5.1.12, this variable was named **Ndb_number_of_storage_nodes**.

- **Not_flushed_delayed_rows**

The number of rows waiting to be written in **INSERT DELAY** queues.

- **Open_files**
The number of files that are open.
- **Open_streams**
The number of streams that are open (used mainly for logging).
- **Open_table_definitions**
The number of cached `.frm` files. This variable was added in MySQL 5.1.3.
- **Open_tables**
The number of tables that are open.
- **Opened_tables**
The number of tables that have been opened. If `Opened_tables` is big, your `table_open_cache` value is probably too small.
- **Prepared_STATEMENT_count**
The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_STATEMENT_count` system variable.) This variable was added in MySQL 5.1.14.
- **Qcache_free_blocks**
The number of free memory blocks in the query cache.
- **Qcache_free_memory**
The amount of free memory for the query cache.
- **Qcache_hits**
The number of query cache hits.
- **Qcache_inserts**
The number of queries added to the query cache.
- **Qcache_lowmem_prunes**
The number of queries that were deleted from the query cache because of low memory.
- **Qcache_not_cached**
The number of non-cached queries (not cacheable, or not cached due to the `query_cache_type` setting).
- **Qcache_queries_in_cache**
The number of queries registered in the query cache.
- **Qcache_total_blocks**
The total number of blocks in the query cache.
- **Questions**
The number of statements that clients have sent to the server.
- **Rpl_status**
The status of fail-safe replication (not yet implemented).
- **Select_full_join**

The number of joins that perform table scans because they do not use indexes. If this value is not 0, you should carefully check the indexes of your tables.

- [Select_full_range_join](#)

The number of joins that used a range search on a reference table.

- [Select_range](#)

The number of joins that used ranges on the first table. This is normally not a critical issue even if the value is quite large.

- [Select_range_check](#)

The number of joins without keys that check for key usage after each row. If this is not 0, you should carefully check the indexes of your tables.

- [Select_scan](#)

The number of joins that did a full scan of the first table.

- [Slave_open_temp_tables](#)

The number of temporary tables that the slave SQL thread currently has open.

- [Slave_running](#)

This is **ON** if this server is a slave that is connected to a master.

- [Slave_retried_transactions](#)

The total number of times since startup that the replication slave SQL thread has retried transactions.

- [Slow_launch_threads](#)

The number of threads that have taken more than [slow_launch_time](#) seconds to create.

- [Slow_queries](#)

The number of queries that have taken more than [long_query_time](#) seconds. See [Section 5.11.5, “The Slow Query Log”](#).

- [Sort_merge_passes](#)

The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the [sort_buffer_size](#) system variable.

- [Sort_range](#)

The number of sorts that were done using ranges.

- [Sort_rows](#)

The number of sorted rows.

- [Sort_scan](#)

The number of sorts that were done by scanning the table.

- [Ssl_xxx](#)

Variables used for SSL connections.

- [Table_locks_immediate](#)

The number of times that a table lock was acquired immediately.

- [Table_locks_waited](#)

The number of times that a table lock could not be acquired immediately and a wait was needed. If this is high and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication.

- [Tc_log_max_pages_used](#)

For the memory-mapped implementation of the log that is used by `mysqld` when it acts as the transaction coordinator for recovery of internal XA transactions, this variable indicates the largest number of pages used for the log since the server started. If the product of [Tc_log_max_pages_used](#) and [Tc_log_page_size](#) is always significantly less than the log size, the size is larger than necessary and can be reduced. (The size is set by the `--log-tc-size` option. Currently, this variable is unused: It is unneeded for binary log-based recovery, and the memory-mapped recovery log method is not used unless the number of storage engines capable of two-phase commit is greater than one. ([InnoDB](#) is the only applicable engine.)

- [Tc_log_page_size](#)

The page size used for the memory-mapped implementation of the XA recovery log. The default value is determined using `get_pagesize()`. Currently, this variable is unused for the same reasons as described for [Tc_log_max_pages_used](#).

- [Tc_log_page_waits](#)

For the memory-mapped implementation of the recovery log, this variable increments each time the server was not able to commit a transaction and had to wait for a free page in the log. If this value is large, you might want to increase the log size (with the `--log-tc-size` option). For binary log-based recovery, this variable increments each time the binary log cannot be closed because there are two-phase commits in progress. (The close operation waits until all such transactions are finished.)

- [Threads_cached](#)

The number of threads in the thread cache.

- [Threads_connected](#)

The number of currently open connections.

- [Threads_created](#)

The number of threads created to handle connections. If [Threads_created](#) is big, you may want to increase the [thread_cache_size](#) value. The cache miss rate can be calculated as [Threads_created/Connections](#).

- [Threads_running](#)

The number of threads that are not sleeping.

- [Uptime](#)

The number of seconds that the server has been up.

5.2.6. SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients. This capability enables each application to tailor the server's operating mode to its own requirements.

For answers to some questions that are often asked about server SQL modes in MySQL, see [Section A.3, “MySQL 5.1 FAQ — Server SQL Mode”](#).

Modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

You can set the default SQL mode by starting `mysqld` with the `--sql-mode="modes"` option, or by using `sql-mode="modes"` in `my.cnf` (Unix operating systems) or `my.ini` (Windows). *modes* is a list of different modes separated by comma (",") characters. The default value is empty (no modes set). The *modes* value also can be empty (`--sql-mode=""` on the command line, or `sql-mode=""` in `my.cnf` on Unix systems or in `my.ini` on Windows) if you want to clear it explicitly.

You can change the SQL mode at runtime by using a `SET [GLOBAL|SESSION] sql_mode='modes'` statement to set the `sql_mode` system value. Setting the `GLOBAL` variable requires the `SUPER` privilege and affects the operation of all clients that connect from that time on. Setting the `SESSION` variable affects only the current client. Any client can change its own session `sql_mode`

value at any time.

Important

SQL mode and user-defined partitioning. Changing the server SQL mode after creating and inserting data into partitioned tables can cause major changes in the behavior of such tables, and could lead to loss or corruption of data. It is strongly recommended that you never change the SQL mode once you have created tables employing user-defined partitioning.

See [Section 17.5, “Restrictions and Limitations on Partitioning”](#), for more information.

You can retrieve the current global or session `sql_mode` value with the following statements:

```
SELECT @@global.sql_mode;  
SELECT @@session.sql_mode;
```

The most important `sql_mode` values are probably these:

- **ANSI**

This mode changes syntax and behavior to conform more closely to standard SQL.

- **STRICT_TRANS_TABLES**

If a value could not be inserted as given into a transactional table, abort the statement. For a non-transactional table, abort the statement if the value occurs in a single-row statement or the first row of a multiple-row statement. More detail is given later in this section.

- **TRADITIONAL**

Make MySQL behave like a “traditional” SQL database system. A simple description of this mode is “give an error instead of a warning” when inserting an incorrect value into a column. **Note:** The `INSERT/UPDATE` aborts as soon as the error is noticed. This may not be what you want if you are using a non-transactional storage engine, because data changes made prior to the error may not be rolled back, resulting in a “partially done” update.

When this manual refers to “strict mode,” it means a mode where at least one of `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` is enabled.

The following list describes all supported modes:

- **ALLOW_INVALID_DATES**

Don't do full checking of dates. Check only that the month is in the range from 1 to 12 and the day is in the range from 1 to 31. This is very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation). This mode applies to `DATE` and `DATETIME` columns. It does not apply to `TIMESTAMP` columns, which always require a valid date.

The server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as `'2004-04-31'` are converted to `'0000-00-00'` and a warning is generated. With strict mode enabled, invalid dates generate an error. To allow such dates, enable `ALLOW_INVALID_DATES`.

- **ANSI_QUOTES**

Treat `'` as an identifier quote character (like the ``` quote character) and not as a string quote character. You can still use ``` to quote identifiers with this mode enabled. With `ANSI_QUOTES` enabled, you cannot use double quotes to quote literal strings, because it is interpreted as an identifier.

- **ERROR_FOR_DIVISION_BY_ZERO**

Produce an error in strict mode (otherwise a warning) when a division by zero (or `MOD(X, 0)`) occurs during an `INSERT` or `UPDATE`. If this mode is not enabled, MySQL instead returns `NULL` for divisions by zero. For `INSERT IGNORE` or `UPDATE IGNORE`, MySQL generates a warning for divisions by zero, but the result of the operation is `NULL`.

- **HIGH_NOT_PRECEDENCE**

The precedence of the `NOT` operator is such that expressions such as `NOT a BETWEEN b AND c` are parsed as `NOT (a BETWEEN b AND c)`. In some older versions of MySQL, the expression was parsed as `(NOT a) BETWEEN b AND c`. The old higher-precedence behavior can be obtained by enabling the `HIGH_NOT_PRECEDENCE` SQL mode.

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
-> 0
mysql> SET sql_mode = 'HIGH_NOT_PRECEDENCE';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
-> 1
```

- **IGNORE_SPACE**

Allow spaces between a function name and the `(` character. This causes built-in function names to be treated as reserved words. As a result, identifiers that are the same as function names must be quoted as described in [Section 8.2, “Identifiers”](#). For example, because there is a `COUNT()` function, the use of `count` as a table name in the following statement causes an error:

```
mysql> CREATE TABLE count (i INT);
ERROR 1064 (42000): You have an error in your SQL syntax
```

The table name should be quoted:

```
mysql> CREATE TABLE `count` (i INT);
Query OK, 0 rows affected (0.00 sec)
```

The `IGNORE_SPACE` SQL mode applies to built-in functions, not to user-defined functions or stored functions. It is always allowable to have spaces after a UDF or stored function name, regardless of whether `IGNORE_SPACE` is enabled.

For further discussion of `IGNORE_SPACE`, see [Section 8.2.4, “Function Name Parsing and Resolution”](#).

- **NO_AUTO_CREATE_USER**

Prevent the `GRANT` statement from automatically creating new users if it would otherwise do so, unless a non-empty password also is specified.

- **NO_AUTO_VALUE_ON_ZERO**

`NO_AUTO_VALUE_ON_ZERO` affects handling of `AUTO_INCREMENT` columns. Normally, you generate the next sequence number for the column by inserting either `NULL` or `0` into it. `NO_AUTO_VALUE_ON_ZERO` suppresses this behavior for `0` so that only `NULL` generates the next sequence number.

This mode can be useful if `0` has been stored in a table's `AUTO_INCREMENT` column. (Storing `0` is not a recommended practice, by the way.) For example, if you dump the table with `mysqldump` and then reload it, MySQL normally generates new sequence numbers when it encounters the `0` values, resulting in a table with contents different from the one that was dumped. Enabling `NO_AUTO_VALUE_ON_ZERO` before reloading the dump file solves this problem. `mysqldump` now automatically includes in its output a statement that enables `NO_AUTO_VALUE_ON_ZERO`, to avoid this problem.

- **NO_BACKSLASH_ESCAPES**

Disable the use of the backslash character (`\`) as an escape character within strings. With this mode enabled, backslash becomes an ordinary character like any other.

- **NO_DIR_IN_CREATE**

When creating a table, ignore all `INDEX DIRECTORY` and `DATA DIRECTORY` directives. This option is useful on slave replication servers.

- **NO_ENGINE_SUBSTITUTION**

Control automatic substitution of the default storage engine when a statement such as `CREATE TABLE` or `ALTER TABLE` specifies a storage engine that is disabled or not compiled in.

Up through MySQL 5.1.11, with `NO_ENGINE_SUBSTITUTION` disabled, the default engine is used and a warning occurs if the desired engine is known but disabled or not compiled in. If the desired engine is invalid (not a known engine name), an error occurs and the table is not created or altered.

With `NO_ENGINE_SUBSTITUTION` enabled, an error occurs and the table is not created or altered if the desired engine is unavailable for any reason (whether disabled or invalid).

As of MySQL 5.1.12, storage engines can be pluggable at runtime, so the distinction between disabled and invalid no longer applies. All unavailable engines are treated the same way:

With `NO_ENGINE_SUBSTITUTION` disabled, for `CREATE TABLE` the default engine is used and a warning occurs if the desired engine is unavailable. For `ALTER TABLE`, a warning occurs and the table is not altered.

With `NO_ENGINE_SUBSTITUTION` enabled, an error occurs and the table is not created or altered if the desired engine is unavailable.

- `NO_FIELD_OPTIONS`

Do not print MySQL-specific column options in the output of `SHOW CREATE TABLE`. This mode is used by `mysqldump` in portability mode.

- `NO_KEY_OPTIONS`

Do not print MySQL-specific index options in the output of `SHOW CREATE TABLE`. This mode is used by `mysqldump` in portability mode.

- `NO_TABLE_OPTIONS`

Do not print MySQL-specific table options (such as `ENGINE`) in the output of `SHOW CREATE TABLE`. This mode is used by `mysqldump` in portability mode.

- `NO_UNSIGNED_SUBTRACTION`

In integer subtraction operations, do not mark the result as `UNSIGNED` if one of the operands is unsigned. In other words, *the result of a subtraction is always signed whenever this mode is in effect, even if one of the operands is unsigned*. For example, compare the type of column `c2` in table `t1` with that of column `c2` in table `t2`:

```
mysql> SET SQL_MODE='';
mysql> CREATE TABLE test (c1 BIGINT UNSIGNED NOT NULL);
mysql> CREATE TABLE t1 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t1;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| c2    | bigint(21) unsigned |      |      | 0        |       |
+-----+-----+-----+-----+-----+-----+

mysql> SET SQL_MODE='NO UNSIGNED SUBTRACTION';
mysql> CREATE TABLE t2 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t2;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| c2    | bigint(21)    |      |      | 0        |       |
+-----+-----+-----+-----+-----+-----+
```

Note that this means that `BIGINT UNSIGNED` is not 100% usable in all contexts. See [Section 11.9, “Cast Functions and Operators”](#).

```
mysql> SET SQL_MODE = '';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
| 18446744073709551615 |
+-----+

mysql> SET SQL_MODE = 'NO UNSIGNED SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
| -1 |
+-----+
```

- `NO_ZERO_DATE`

In strict mode, don't allow '0000-00-00' as a valid date. You can still insert zero dates with the [IGNORE](#) option. When not in strict mode, the date is accepted but a warning is generated.

- [NO_ZERO_IN_DATE](#)

In strict mode, don't accept dates where the month or day part is 0. If used with the [IGNORE](#) option, MySQL inserts a '0000-00-00' date for any such date. When not in strict mode, the date is accepted but a warning is generated.

- [ONLY_FULL_GROUP_BY](#)

Do not allow queries for which the [SELECT](#) list refers to non-aggregated columns that are not named in the [GROUP BY](#) clause. The following query is invalid with this mode enabled because [address](#) is not named in the [GROUP BY](#) clause:

```
SELECT name, address, MAX(age) FROM t GROUP BY name;
```

As of MySQL 5.1.11, this mode also restricts references to non-aggregated columns in the [HAVING](#) clause that are not named in the [GROUP BY](#) clause.

- [PIPES_AS_CONCAT](#)

Treat `||` as a string concatenation operator (same as [CONCAT\(\)](#)) rather than as a synonym for [OR](#).

- [REAL_AS_FLOAT](#)

Treat [REAL](#) as a synonym for [FLOAT](#). By default, MySQL treats [REAL](#) as a synonym for [DOUBLE](#).

- [STRICT_ALL_TABLES](#)

Enable strict mode for all storage engines. Invalid data values are rejected. Additional detail follows.

- [STRICT_TRANS_TABLES](#)

Enable strict mode for transactional storage engines, and when possible for non-transactional storage engines. Additional details follow.

Strict mode controls how MySQL handles input values that are invalid or missing. A value can be invalid for several reasons. For example, it might have the wrong data type for the column, or it might be out of range. A value is missing when a new row to be inserted does not contain a value for a non-[NULL](#) column that has no explicit [DEFAULT](#) clause in its definition. (For a [NULL](#) column, [NULL](#) is inserted if the value is missing.)

For transactional tables, an error occurs for invalid or missing values in a statement when either of the [STRICT_ALL_TABLES](#) or [STRICT_TRANS_TABLES](#) modes are enabled. The statement is aborted and rolled back.

For non-transactional tables, the behavior is the same for either mode, if the bad value occurs in the first row to be inserted or updated. The statement is aborted and the table remains unchanged. If the statement inserts or modifies multiple rows and the bad value occurs in the second or later row, the result depends on which strict option is enabled:

- For [STRICT_ALL_TABLES](#), MySQL returns an error and ignores the rest of the rows. However, in this case, the earlier rows still have been inserted or updated. This means that you might get a partial update, which might not be what you want. To avoid this, it's best to use single-row statements because these can be aborted without changing the table.
- For [STRICT_TRANS_TABLES](#), MySQL converts an invalid value to the closest valid value for the column and insert the adjusted value. If a value is missing, MySQL inserts the implicit default value for the column data type. In either case, MySQL generates a warning rather than an error and continues processing the statement. Implicit defaults are described in [Section 10.1.4, “Data Type Default Values”](#).

Strict mode disallows invalid date values such as '2004-04-31'. It does not disallow dates with zero parts such as '2004-04-00' or “zero” dates. To disallow these as well, enable the [NO_ZERO_IN_DATE](#) and [NO_ZERO_DATE](#) SQL modes in addition to strict mode.

If you are not using strict mode (that is, neither [STRICT_TRANS_TABLES](#) nor [STRICT_ALL_TABLES](#) is enabled), MySQL inserts adjusted values for invalid or missing values and produces warnings. In strict mode, you can produce this behavior by using [INSERT](#)

IGNORE or **UPDATE IGNORE**. See [Section 12.5.4.32, “SHOW WARNINGS Syntax”](#).

The following special modes are provided as shorthand for combinations of mode values from the preceding list.

The descriptions include all mode values that are available in the most recent version of MySQL. For older versions, a combination mode does not include individual mode values that are not available except in newer versions.

- **ANSI**

Equivalent to **REAL_AS_FLOAT**, **PIPES_AS_CONCAT**, **ANSI_QUOTES**, **IGNORE_SPACE**.

As of MySQL 5.1.18, **ANSI** mode also causes the server to return an error for queries where a set function *S* with an outer reference *S(outer_ref)* cannot be aggregated in the outer query against which the outer reference has been resolved. This is such a query:

```
SELECT * FROM t1 WHERE t1.a IN (SELECT MAX(t1.b) FROM t2 WHERE ...);
```

Here, **MAX(t1.b)** cannot be aggregated in the outer query because it appears in the **WHERE** clause of that query. Standard SQL requires an error in this situation. If **ANSI** mode is not enabled, the server treats *S(outer_ref)* in such queries the same way that it would interpret *S(const)*, as was always done prior to 5.1.18.

See [Section 1.9.3, “Running MySQL in ANSI Mode”](#).

- **DB2**

Equivalent to **PIPES_AS_CONCAT**, **ANSI_QUOTES**, **IGNORE_SPACE**, **NO_KEY_OPTIONS**, **NO_TABLE_OPTIONS**, **NO_FIELD_OPTIONS**.

- **MAXDB**

Equivalent to **PIPES_AS_CONCAT**, **ANSI_QUOTES**, **IGNORE_SPACE**, **NO_KEY_OPTIONS**, **NO_TABLE_OPTIONS**, **NO_FIELD_OPTIONS**, **NO_AUTO_CREATE_USER**.

- **MSSQL**

Equivalent to **PIPES_AS_CONCAT**, **ANSI_QUOTES**, **IGNORE_SPACE**, **NO_KEY_OPTIONS**, **NO_TABLE_OPTIONS**, **NO_FIELD_OPTIONS**.

- **MYSQL323**

Equivalent to **NO_FIELD_OPTIONS**, **HIGH_NOT_PRECEDENCE**.

- **MYSQL40**

Equivalent to **NO_FIELD_OPTIONS**, **HIGH_NOT_PRECEDENCE**.

- **ORACLE**

Equivalent to **PIPES_AS_CONCAT**, **ANSI_QUOTES**, **IGNORE_SPACE**, **NO_KEY_OPTIONS**, **NO_TABLE_OPTIONS**, **NO_FIELD_OPTIONS**, **NO_AUTO_CREATE_USER**.

- **POSTGRESQL**

Equivalent to **PIPES_AS_CONCAT**, **ANSI_QUOTES**, **IGNORE_SPACE**, **NO_KEY_OPTIONS**, **NO_TABLE_OPTIONS**, **NO_FIELD_OPTIONS**.

- **TRADITIONAL**

Equivalent to **STRICT_TRANS_TABLES**, **STRICT_ALL_TABLES**, **NO_ZERO_IN_DATE**, **NO_ZERO_DATE**, **ERROR_FOR_DIVISION_BY_ZERO**, **NO_AUTO_CREATE_USER**.

5.2.7. The Shutdown Process

The server shutdown process takes place as follows:

1. The shutdown process is initiated.

Server shutdown can be initiated several ways. For example, a user with the `SHUTDOWN` privilege can execute a `mysqladmin shutdown` command. `mysqladmin` can be used on any platform supported by MySQL. Other operating system-specific shutdown initiation methods are possible as well: The server shuts down on Unix when it receives a `SIGTERM` signal. A server running as a service on Windows shuts down when the services manager tells it to.

2. The server creates a shutdown thread if necessary.

Depending on how shutdown was initiated, the server might create a thread to handle the shutdown process. If shutdown was requested by a client, a shutdown thread is created. If shutdown is the result of receiving a `SIGTERM` signal, the signal thread might handle shutdown itself, or it might create a separate thread to do so. If the server tries to create a shutdown thread and cannot (for example, if memory is exhausted), it issues a diagnostic message that appears in the error log:

```
Error: Can't create thread to kill server
```

3. The server stops accepting new connections.

To prevent new activity from being initiated during shutdown, the server stops accepting new client connections. It does this by closing the network connections to which it normally listens for connections: the TCP/IP port, the Unix socket file, the Windows named pipe, and shared memory on Windows.

4. The server terminates current activity.

For each thread that is associated with a client connection, the connection to the client is broken and the thread is marked as killed. Threads die when they notice that they are so marked. Threads for idle connections die quickly. Threads that currently are processing statements check their state periodically and take longer to die. For additional information about thread termination, see [Section 12.5.5.3, “KILL Syntax”](#), in particular for the instructions about killed `REPAIR TABLE` or `OPTIMIZE TABLE` operations on `MyISAM` tables.

For threads that have an open transaction, the transaction is rolled back. Note that if a thread is updating a non-transactional table, an operation such as a multiple-row `UPDATE` or `INSERT` may leave the table partially updated, because the operation can terminate before completion.

If the server is a master replication server, threads associated with currently connected slaves are treated like other client threads. That is, each one is marked as killed and exits when it next checks its state.

If the server is a slave replication server, the I/O and SQL threads, if active, are stopped before client threads are marked as killed. The SQL thread is allowed to finish its current statement (to avoid causing replication problems), and then stops. If the SQL thread was in the middle of a transaction at this point, the transaction is rolled back.

5. Storage engines are shut down or closed.

At this stage, the table cache is flushed and all open tables are closed.

Each storage engine performs any actions necessary for tables that it manages. For example, `MyISAM` flushes any pending index writes for a table. `InnoDB` flushes its buffer pool to disk, unless `innodb_fast_shutdown` is 2, writes the current LSN to the tablespace, and terminates its own internal threads.

6. The server exits.

5.2.8. Server-Side Help

MySQL Server supports a `HELP` statement that returns online information from the MySQL Reference manual (see [Section 12.3.2, “HELP Syntax”](#)). The proper operation of this statement requires that the help tables in the `mysql` database be initialized with help topic information, which is done by processing the contents of the `fill_help_tables.sql` script.

For a MySQL binary distribution on Unix, help table setup occurs when you run `mysql_install_db`. For an RPM distribution on Linux or binary distribution on Windows, help table setup occurs as part of the MySQL installation process.

For a MySQL source distribution, you can find the `fill_help_tables.sql` file in the `scripts` directory. To load the file manually, make sure that you have initialized the `mysql` database by running `mysql_install_db`, and then process the file with the `mysql` client as follows:

```
shell> mysql -u root mysql < fill_help_tables.sql
```

If you are working with BitKeeper and a MySQL development source tree, the tree doesn't contain `fill_help_tables.sql`. You can download the proper file for your version of MySQL from <http://dev.mysql.com/doc/>. After downloading and uncompressing the file, process it with `mysql` as just described.

5.3. MySQL Server Startup Programs

This section describes several programs that are used to start `mysqld`, the MySQL server.

5.3.1. `mysqld_safe` — MySQL Server Startup Script

`mysqld_safe` is the recommended way to start a `mysqld` server on Unix and NetWare. `mysqld_safe` adds some safety features such as restarting the server when an error occurs and logging runtime information to an error log file. Descriptions of error logging and NetWare-specific behaviors are given later in this section.

Note

In MySQL 5.1.20 (only), the default error logging behavior with `mysqld_safe` is to write errors to `syslog` on systems that support the `logger` program. This differs from the default behavior of writing an error log file for other versions.

In 5.1.20, logging to `syslog` may fail to operate correctly in some cases, so we recommend that you use `--skip-syslog` to use the default log file or `--log-error=file_name` to specify a log filename explicitly.

`mysqld_safe` tries to start an executable named `mysqld`. To override the default behavior and specify explicitly the name of the server you want to run, specify a `--mysqld` or `--mysqld-version` option to `mysqld_safe`. You can also use `--ledir` to indicate the directory where `mysqld_safe` should look for the server.

Many of the options to `mysqld_safe` are the same as the options to `mysqld`. See [Section 5.2.2, “Command Options”](#).

All options specified to `mysqld_safe` on the command line are passed to `mysqld`. If you want to use any options that are specific to `mysqld_safe` and that `mysqld` doesn't support, do not specify them on the command line. Instead, list them in the `[mysqld_safe]` group of an option file. See [Section 4.3.2, “Using Option Files”](#).

`mysqld_safe` reads all options from the `[mysqld]`, `[server]`, and `[mysqld_safe]` sections in option files. For example, if you specify a `[mysqld]` section like this, `mysqld_safe` will find and use the `--log-error` option:

```
[mysqld]
log-error=error.log
```

For backward compatibility, `mysqld_safe` also reads `[safe_mysqld]` sections, although you should rename such sections to `[mysqld_safe]` in MySQL 5.1 installations.

`mysqld_safe` supports the following options:

- `--help`

Display a help message and exit.

- `--autoclose`

(NetWare only) On NetWare, `mysqld_safe` provides a screen presence. When you unload (shut down) the `mysqld_safe` NLM, the screen does not by default go away. Instead, it prompts for user input:

```
*<NLM has terminated; Press any key to close the screen>*
```

If you want NetWare to close the screen automatically instead, use the `--autoclose` option to `mysqld_safe`.

- `--basedir=path`

The path to the MySQL installation directory.

- `--core-file-size=size`

The size of the core file that `mysqld` should be able to create. The option value is passed to `ulimit -c`.

- `--datadir=path`

The path to the data directory.

- `--defaults-extra-file=path`

The name of an option file to be read in addition to the usual option files. This must be the first option on the command line if it is used. If the file does not exist or is otherwise inaccessible, the server will exit with an error.

- `--defaults-file=file_name`

The name of an option file to be read instead of the usual option files. This must be the first option on the command line if it is used.

- `--ledir=path`

If `mysqld_safe` cannot find the server, use this option to indicate the pathname to the directory where the server is located.

- `--log-error=file_name`

Write the error log to the given file. See [Section 5.11.2, “The Error Log”](#).

- `--mysqld=prog_name`

The name of the server program (in the `ledir` directory) that you want to start. This option is needed if you use the MySQL binary distribution but have the data directory outside of the binary distribution. If `mysqld_safe` cannot find the server, use the `--ledir` option to indicate the pathname to the directory where the server is located.

- `--mysqld-version=suffix`

This option is similar to the `--mysqld` option, but you specify only the suffix for the server program name. The basename is assumed to be `mysqld`. For example, if you use `--mysqld-version=debug`, `mysqld_safe` starts the `mysqld-debug` program in the `ledir` directory. If the argument to `--mysqld-version` is empty, `mysqld_safe` uses `mysqld` in the `ledir` directory.

- `--nice=priority`

Use the `nice` program to set the server's scheduling priority to the given value.

- `--no-defaults`

Do not read any option files. This must be the first option on the command line if it is used.

- `--open-files-limit=count`

The number of files that `mysqld` should be able to open. The option value is passed to `ulimit -n`. Note that you need to start `mysqld_safe` as `root` for this to work properly!

- `--pid-file=file_name`

The pathname of the process ID file.

- `--port=port_num`

The port number that the server should use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the `root` system user.

- `--skip-kill-mysqld`

Do not try to kill stray `mysqld` processes at startup. This option works only on Linux.

- `--socket=path`

The Unix socket file that the server should use when listening for local connections.

- `--syslog`, `--skip-syslog`

`--syslog` causes error messages to be sent to `syslog` on systems that support the `logger` program. `--skip-syslog` suppresses the use of `syslog`; messages are written to an error log file. These options were added in MySQL 5.1.20.

- `--syslog-tag=tag`

For logging to `syslog`, messages from `mysqld_safe` and `mysqld` are written with a tag of `mysqld_safe` and `mysqld`, respectively. To specify a suffix for the tag, use `--syslog-tag=tag`, which modifies the tags to be `mysqld_safe-tag` and `mysqld-tag`. This option was added in MySQL 5.1.21.

- `--timezone=timezone`

Set the `TZ` time zone environment variable to the given option value. Consult your operating system documentation for legal time zone specification formats.

- `--user={user_name|user_id}`

Run the `mysqld` server as the user having the name `user_name` or the numeric user ID `user_id`. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

If you execute `mysqld_safe` with the `--defaults-file` or `--defaults-extra-option` option to name an option file, the option must be the first one given on the command line or the option file will not be used. For example, this command will not use the named option file:

```
mysql> mysqld_safe --port=port_num --defaults-file=file_name
```

Instead, use the following command:

```
mysql> mysqld_safe --defaults-file=file_name --port=port_num
```

The `mysqld_safe` script is written so that it normally can start a server that was installed from either a source or a binary distribution of MySQL, even though these types of distributions typically install the server in slightly different locations. (See [Section 2.1.5](#), “Installation Layouts”.) `mysqld_safe` expects one of the following conditions to be true:

- The server and databases can be found relative to the working directory (the directory from which `mysqld_safe` is invoked). For binary distributions, `mysqld_safe` looks under its working directory for `bin` and `data` directories. For source distributions, it looks for `libexec` and `var` directories. This condition should be met if you execute `mysqld_safe` from your MySQL installation directory (for example, `/usr/local/mysql` for a binary distribution).
- If the server and databases cannot be found relative to the working directory, `mysqld_safe` attempts to locate them by absolute pathnames. Typical locations are `/usr/local/libexec` and `/usr/local/var`. The actual locations are determined from the values configured into the distribution at the time it was built. They should be correct if MySQL is installed in the location specified at configuration time.

Because `mysqld_safe` tries to find the server and databases relative to its own working directory, you can install a binary distribution of MySQL anywhere, as long as you run `mysqld_safe` from the MySQL installation directory:

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

If `mysqld_safe` fails, even when invoked from the MySQL installation directory, you can specify the `--ledir` and `--datadir` options to indicate the directories in which the server and databases are located on your system.

When you use `mysqld_safe` to start `mysqld`, `mysqld_safe` arranges for error (and notice) messages from itself and from `mysqld` to go to the same destination.

As of MySQL 5.1.20, there are several `mysqld_safe` options for controlling the destination of these messages:

- `--syslog`: Write error messages to `syslog` on systems that support the `logger` program.

- `--skip-syslog`: Do not write error messages to `syslog`. Messages are written to the default error log file (`host_name.err` in the data directory), or to a named file if the `--log-error` option is given.
- `--log-error=file_name`: Write error messages to the named error file.

If none of these options is given, the default is `--skip-syslog`.

Note

In MySQL 5.1.20 [only](#), the default is `--syslog`. This differs from logging behavior for other versions of MySQL, for which the default is to write messages to the default error log file.

If `--syslog` and `--log-error` are both given, a warning is issued and `--log-error` takes precedence.

When `mysqld_safe` writes a message, notices go to the logging destination (`syslog` or the error log file) and `stdout`. Errors go to the logging destination and `stderr`.

Before MySQL 5.1.20, error logging is controlled only with the `--log-error` option. If it is given, messages go to the named error file. Otherwise, messages go to the default error file.

Normally, you should not edit the `mysqld_safe` script. Instead, configure `mysqld_safe` by using command-line options or options in the `[mysqld_safe]` section of a `my.cnf` option file. In rare cases, it might be necessary to edit `mysqld_safe` to get it to start the server properly. However, if you do this, your modified version of `mysqld_safe` might be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.

On NetWare, `mysqld_safe` is a NetWare Loadable Module (NLM) that is ported from the original Unix shell script. It starts the server as follows:

1. Runs a number of system and option checks.
2. Runs a check on `MyISAM` tables.
3. Provides a screen presence for the MySQL server.
4. Starts `mysqld`, monitors it, and restarts it if it terminates in error.
5. Sends error messages from `mysqld` to the `host_name.err` file in the data directory.
6. Sends `mysqld_safe` screen output to the `host_name.safe` file in the data directory.

5.3.2. `mysql.server` — MySQL Server Startup Script

MySQL distributions on Unix include a script named `mysql.server`. It can be used on systems such as Linux and Solaris that use System V-style run directories to start and stop system services. It is also used by the Mac OS X Startup Item for MySQL.

`mysql.server` can be found in the `support-files` directory under your MySQL installation directory or in a MySQL source distribution.

If you use the Linux server RPM package (`MySQL-server-VERSION.rpm`), the `mysql.server` script will be installed in the `/etc/init.d` directory with the name `mysql`. You need not install it manually. See [Section 2.4, “Installing MySQL from RPM Packages on Linux”](#), for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as `mysqld`.

If you install MySQL from a source distribution or using a binary distribution format that does not install `mysql.server` automatically, you can install it manually. Instructions are provided in [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).

`mysql.server` reads options from the `[mysql.server]` and `[mysqld]` sections of option files. For backward compatibility, it also reads `[mysql_server]` sections, although you should rename such sections to `[mysql.server]` when using MySQL 5.1.

`mysql.server` understands the following options:

- `--basedir=path`

The path to the MySQL installation directory.

- `--datadir=path`

The path to the MySQL data directory.

- `--pid-file=file_name`

The pathname of the file in which the server should write its process ID.

- `--service-startup-timeout=file_name`

How long in seconds to wait for confirmation of server startup. If the server does not start within this time, `mysql.server` exits with an error. The default value is 900. A value of 0 means not to wait at all for startup. Negative values mean to wait forever (no timeout). This option was added in MySQL 5.1.17. Before that, a value of 900 is always used.

- `--use-mysqld_safe`

Use `mysqld_safe` to start the server. This is the default.

- `--use-manager`

Use Instance Manager to start the server.

- `--user=user_name`

The login username to use for running `mysqld`.

5.3.3. `mysqld_multi` — Manage Multiple MySQL Servers

`mysqld_multi` is designed to manage several `mysqld` processes that listen for connections on different Unix socket files and TCP/IP ports. It can start or stop servers, or report their current status. The MySQL Instance Manager is an alternative means of managing multiple servers (see [Section 5.4, “mysqlmanager — The MySQL Instance Manager”](#)).

`mysqld_multi` searches for groups named `[mysqldN]` in `my.cnf` (or in the file named by the `--config-file` option). `N` can be any positive integer. This number is referred to in the following discussion as the option group number, or *GNR*. Group numbers distinguish option groups from one another and are used as arguments to `mysqld_multi` to specify which servers you want to start, stop, or obtain a status report for. Options listed in these groups are the same that you would use in the `[mysqld]` group used for starting `mysqld`. (See, for example, [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).) However, when using multiple servers, it is necessary that each one use its own value for options such as the Unix socket file and TCP/IP port number. For more information on which options must be unique per server in a multiple-server environment, see [Section 5.12, “Running Multiple MySQL Servers on the Same Machine”](#).

To invoke `mysqld_multi`, use the following syntax:

```
shell> mysqld_multi [options] {start|stop|report} [GNR[,GNR] ...]
```

`start`, `stop`, and `report` indicate which operation to perform. You can perform the designated operation for a single server or multiple servers, depending on the *GNR* list that follows the option name. If there is no list, `mysqld_multi` performs the operation for all servers in the option file.

Each *GNR* value represents an option group number or range of group numbers. The value should be the number at the end of the group name in the option file. For example, the *GNR* for a group named `[mysqld17]` is `17`. To specify a range of numbers, separate the first and last numbers by a dash. The *GNR* value `10-13` represents groups `[mysqld10]` through `[mysqld13]`. Multiple groups or group ranges can be specified on the command line, separated by commas. There must be no whitespace characters (spaces or tabs) in the *GNR* list; anything after a whitespace character is ignored.

This command starts a single server using option group `[mysqld17]`:

```
shell> mysqld_multi start 17
```

This command stops several servers, using option groups `[mysqld8]` and `[mysqld10]` through `[mysqld13]`:

```
shell> mysqld_multi stop 8,10-13
```

For an example of how you might set up an option file, use this command:

```
shell> mysqld_multi --example
```

As of MySQL 5.1.18, `mysqld_multi` searches for option files as follows:

- With `--no-defaults`, no option files are read.
- With `--defaults-file=file_name`, only the named file is read.
- Otherwise, option files in the standard list of locations are read, including any file named by the `--defaults-extra-file=file_name` option, if one is given. (If the option is given multiple times, the last value is used.)

Option files read are searched for `[mysqld_multi]` and `[mysqldN]` option groups.

Before MySQL 5.1.18, the preceding options are not recognized. Files in the standard locations are read, and any file named by the `--config-file=file_name` option, if one is given. A file named by `--config-file` is read only for `[mysqldN]` option groups, not the `[mysqld_multi]` group.

`mysqld_multi` supports the following options:

- `--help`

Display a help message and exit.

- `--config-file=file_name`

As of MySQL 5.1.18, this option is deprecated. If given, it is treated the same way as `--defaults-extra-file`, described earlier.

Before MySQL 5.1.18, this option specifies the name of an extra option file. It affects where `mysqld_multi` looks for `[mysqldN]` option groups. Without this option, all options are read from the usual `my.cnf` file. The option does not affect where `mysqld_multi` reads its own options, which are always taken from the `[mysqld_multi]` group in the usual `my.cnf` file.

- `--example`

Display a sample option file.

- `--log=file_name`

Specify the name of the log file. If the file exists, log output is appended to it.

- `--mysqladmin=prog_name`

The `mysqladmin` binary to be used to stop servers.

- `--mysqld=prog_name`

The `mysqld` binary to be used. Note that you can specify `mysqld_safe` as the value for this option also. If you use `mysqld_safe` to start the server, you can include the `mysqld` or `ledir` options in the corresponding `[mysqldN]` option group. These options indicate the name of the server that `mysqld_safe` should start and the pathname of the directory where the server is located. (See the descriptions for these options in [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#).) Example:

```
[mysqld38]
mysqld = mysqld-debug
ledir  = /opt/local/mysql/libexec
```

- `--no-log`

Print log information to `stdout` rather than to the log file. By default, output goes to the log file.

- `--password=password`

The password of the MySQL account to use when invoking `mysqladmin`. Note that the password value is not optional for this option, unlike for other MySQL programs.

- `--silent`

Silent mode; disable warnings.

- `--tcp-ip`

Connect to each MySQL server via the TCP/IP port instead of the Unix socket file. (If a socket file is missing, the server might still be running, but accessible only via the TCP/IP port.) By default, connections are made using the Unix socket file. This option affects `stop` and `report` operations.

- `--user=user_name`

The username of the MySQL account to use when invoking `mysqladmin`.

- `--verbose`

Be more verbose.

- `--version`

Display version information and exit.

Some notes about `mysqld_multi`:

- **Most important:** Before using `mysqld_multi` be sure that you understand the meanings of the options that are passed to the `mysqld` servers and *why* you would want to have separate `mysqld` processes. Beware of the dangers of using multiple `mysqld` servers with the same data directory. Use separate data directories, unless you *know* what you are doing. Starting multiple servers with the same data directory does *not* give you extra performance in a threaded system. See [Section 5.12, “Running Multiple MySQL Servers on the Same Machine”](#).

-

Important

Make sure that the data directory for each server is fully accessible to the Unix account that the specific `mysqld` process is started as. *Do not* use the Unix `root` account for this, unless you *know* what you are doing. See [Section 5.6.5, “How to Run MySQL as a Normal User”](#).

- Make sure that the MySQL account used for stopping the `mysqld` servers (with the `mysqladmin` program) has the same username and password for each server. Also, make sure that the account has the `SHUTDOWN` privilege. If the servers that you want to manage have different usernames or passwords for the administrative accounts, you might want to create an account on each server that has the same username and password. For example, you might set up a common `multi_admin` account by executing the following commands for each server:

```
shell> mysql -u root -S /tmp/mysql.sock -p
Enter password:
mysql> GRANT SHUTDOWN ON *.*
-> TO 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
```

See [Section 5.7.2, “How the Privilege System Works”](#). You have to do this for each `mysqld` server. Change the connection parameters appropriately when connecting to each one. Note that the hostname part of the account name must allow you to connect as `multi_admin` from the host where you want to run `mysqld_multi`.

- The Unix socket file and the TCP/IP port number must be different for every `mysqld`. (Alternatively, if the host has multiple network addresses, you can use `--bind-address` to cause different servers to listen to different interfaces.)
- The `--pid-file` option is very important if you are using `mysqld_safe` to start `mysqld` (for example, `-mysqld=mysqld_safe`). Every `mysqld` should have its own process ID file. The advantage of using `mysqld_safe` instead of `mysqld` is that `mysqld_safe` monitors its `mysqld` process and restarts it if the process terminates due to a signal sent using `kill -9` or for other reasons, such as a segmentation fault. Please note that the `mysqld_safe` script might require that you start

it from a certain place. This means that you might have to change location to a certain directory before running `mysqld_multi`. If you have problems starting, please see the `mysqld_safe` script. Check especially the lines:

```
-----
MY_PWD=`pwd`
# Check if we are starting this relative (for the binary release)
if test -d $MY_PWD/data/mysql -a \
  -f ./share/mysql/english/errmsg.sys -a \
  -x ./bin/mysqld
-----
```

The test performed by these lines should be successful, or you might encounter problems. See [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#).

- You might want to use the `--user` option for `mysqld`, but to do this you need to run the `mysqld_multi` script as the Unix `root` user. Having the option in the option file doesn't matter; you just get a warning if you are not the superuser and the `mysqld` processes are started under your own Unix account.

The following example shows how you might set up an option file for use with `mysqld_multi`. The order in which the `mysqld` programs are started or stopped depends on the order in which they appear in the option file. Group numbers need not form an unbroken sequence. The first and fifth `[mysqldN]` groups were intentionally omitted from the example to illustrate that you can have “gaps” in the option file. This gives you more flexibility.

```
# This file should probably be in your home dir (~/.my.cnf)
# or /etc/my.cnf
# Version 2.1 by Jani Tolonen

[mysqld_multi]
mysqld      = /usr/local/bin/mysqld_safe
mysqldadmin = /usr/local/bin/mysqldadmin
user        = multi_admin
password    = multipass

[mysqld2]
socket      = /tmp/mysql.sock2
port        = 3307
pid-file    = /usr/local/mysql/var2/hostname.pid2
datadir     = /usr/local/mysql/var2
language    = /usr/local/share/mysql/english
user        = john

[mysqld3]
socket      = /tmp/mysql.sock3
port        = 3308
pid-file    = /usr/local/mysql/var3/hostname.pid3
datadir     = /usr/local/mysql/var3
language    = /usr/local/share/mysql/swedish
user        = monty

[mysqld4]
socket      = /tmp/mysql.sock4
port        = 3309
pid-file    = /usr/local/mysql/var4/hostname.pid4
datadir     = /usr/local/mysql/var4
language    = /usr/local/share/mysql/estonia
user        = tonu

[mysqld6]
socket      = /tmp/mysql.sock6
port        = 3311
pid-file    = /usr/local/mysql/var6/hostname.pid6
datadir     = /usr/local/mysql/var6
language    = /usr/local/share/mysql/japanese
user        = jani
```

See [Section 4.3.2, “Using Option Files”](#).

5.4. `mysqlmanager` — The MySQL Instance Manager

`mysqlmanager` is the MySQL Instance Manager (IM). This program monitors and manages MySQL Database Server instances. MySQL Instance Manager is available for Unix-like operating systems, as well as Windows. It runs as a daemon that listens on a TCP/IP port. On Unix, it also listens on a Unix socket file.

MySQL Instance Manager can be used in place of the `mysqld_safe` script to start and stop one or more instances of MySQL Server. Because Instance Manager can manage multiple server instances, it can also be used in place of the `mysqld_multi` script. Instance

Manager offers these capabilities:

- Instance Manager can start and stop instances, and report on the status of instances.
- Server instances can be treated as guarded or unguarded:
 - When Instance Manager starts, it starts each guarded instance. If the instance crashes, Instance Manager detects this and restarts it. When Instance Manager stops, it stops the instance.
 - A nonguarded instance is not started when Instance Manager starts or monitored by it. If the instance crashes after being started, Instance Manager does not restart it. When Instance Manager exits, it does not stop the instance if it is running.

Instances are guarded by default. An instance can be designated as nonguarded by including the `nonguarded` option in the configuration file.

- Instance Manager provides an interactive interface for configuring instances, so that the need to edit the configuration file manually is reduced or eliminated.
- Instance Manager provides remote instance management. That is, it runs on the host where you want to control MySQL Server instances, but you can connect to it from a remote host to perform instance-management operations.

The following sections describe MySQL Instance Manager operation in more detail.

5.4.1. MySQL Instance Manager Command Options

The MySQL Instance Manager supports a number of command options. For a brief listing, invoke `mysqlmanager` with the `--help` option. Options may be given on the command line or in the Instance Manager configuration file. On Windows, the standard configuration file is `my.ini` in the directory where Instance Manager is installed. On Unix, the standard file is `/etc/my.cnf`. To specify a different configuration file, start Instance Manager with the `--defaults-file` option.

`mysqlmanager` supports the options described in the following list. The options for managing entries in the password file are described further in [Section 5.4.4, “Instance Manager User and Password Management”](#).

- `--help, -?`

Display a help message and exit.

- `--add-user`

Add a new user (specified with the `--username` option) to the password file. This option was added in MySQL 5.1.12.

- `--angel-pid-file=file_name`

The file in which the angel process records its process ID when `mysqlmanager` runs in daemon mode (that is, when the `--run-as-service` option is given). The default filename is `mysqlmanager.angel.pid`.

If the `--angel-pid-file` option is not given, the default angel PID file has the same name as the PID file except that any PID file extension is replaced with an extension of `.angel.pid`. (For example, `mysqlmanager.pid` becomes `mysqlmanager.angel.pid`.)

This option was added in MySQL 5.1.11.

- `--bind-address=IP`

The IP address to bind to.

- `--check-password-file`

Check the validity and consistency of the password file. This option was added in MySQL 5.1.12.

- `--clean-password-file`

Drop all users from the password file. This option was added in MySQL 5.1.12.

- `--debug=debug_options, -# debug_options`

Write a debugging log. The `debug_options` string often is `'d:t:o,file_name'`. This option was added in MySQL 5.1.10.

- `--default-mysqld-path=path`

The pathname of the MySQL Server binary. This pathname is used for all server instance sections in the configuration file for which no `mysqld-path` option is present. The default value of this option is the compiled-in pathname, which depends on how the MySQL distribution was configured. Example: `--default-mysqld-path=/usr/sbin/mysqld`

- `--defaults-file=file_name`

Read Instance Manager and MySQL Server settings from the given file. All configuration changes made by the Instance Manager will be written to this file. This must be the first option on the command line if it is used, and the file must exist.

If this option is not given, Instance Manager uses its standard configuration file. On Windows, the standard file is `my.ini` in the directory where Instance Manager is installed. On Unix, the standard file is `/etc/my.cnf`.

- `--drop-user`

Drop a user (specified with the `--username` option) from the password file. This option was added in MySQL 5.1.12.

- `--edit-user`

Change an entry for an existing user (specified with the `--username` option) in the password file. This option was added in MySQL 5.1.12.

- `--install`

On Windows, install Instance Manager as a Windows service. The service name is `MySQL Manager`.

- `--list-users`

List the users in the password file. This option was added in MySQL 5.1.12.

- `--log=file_name`

The path to the Instance Manager log file. This option has no effect unless the `--run-as-service` option is also given. If the filename specified for the option is a relative name, the log file is created under the directory from which Instance Manager is started. To ensure that the file is created in a specific directory, specify it as a full pathname.

If `--run-as-service` is given without `--log`, the log file is `mysqlmanager.log` in the data directory.

If `--run-as-service` is not given, log messages go to the standard output. To capture log output, you can redirect Instance Manager output to a file:

```
mysqlmanager > im.log
```

- `--monitoring-interval=seconds`

The interval in seconds for monitoring server instances. The default value is 20 seconds. Instance Manager tries to connect to each monitored (guarded) instance using the non-existing `MySQL_Instance_Manager` user account to check whether it is alive/not hanging. If the result of the connection attempt indicates that the instance is unavailable, Instance Manager performs several attempts to restart the instance.

Normally, the `MySQL_Instance_Manager` account does not exist, so the connection attempts by Instance Manager cause the monitored instance to produce messages in its general query log similar to the following:

```
Access denied for user 'MySQL_Instance_M'@'localhost' »  
(using password: YES)
```

The `nonguarded` option in the appropriate server instance section disables monitoring for a particular instance. If the instance dies after being started, Instance Manager will not restart it. Instance Manager tries to connect to a nonguarded instance only when you request the instance's status (for example, with the `SHOW INSTANCES` status).

See [Section 5.4.5, “MySQL Server Instance Status Monitoring”](#), for more information.

- `--mysqld-safe-compatible`

Run in a `mysqld_safe`-compatible manner. For details, see [Section 5.4.3, “Starting the MySQL Server with MySQL Instance Manager”](#). This option was added in MySQL 5.1.12.

- `--password=password, -p password`

Specify the password for an entry to be added to or modified in the password file. Unlike the `--password/-P` option for most MySQL programs, the password value is required, not optional. This option was added in MySQL 5.1.12.

- `--password-file=file_name`

The name of the file where the Instance Manager looks for users and passwords. On Windows, the default is `mysqlmanager.passwd` in the directory where Instance Manager is installed. On Unix, the default file is `/etc/mysqlmanager.passwd`.

- `--pid-file=file_name`

The process ID file to use. On Windows, the default file is `mysqlmanager.pid` in the directory where Instance Manager is installed. On Unix, the default is `mysqlmanager.pid` in the data directory.

- `--port=port_num`

The port number to use when listening for TCP/IP connections from clients. The default port number (assigned by IANA) is 2273.

- `--print-defaults`

Print the current defaults and exit. This must be the first option on the command line if it is used.

- `--print-password-line`

Prepare an entry for the password file, print it to the standard output, and exit. You can redirect the output from Instance Manager to a file to save the entry in the file.

Prior to MySQL 5.1.12, this option was named `--passwd`.

- `--remove`

On Windows, removes Instance Manager as a Windows service. This assumes that Instance Manager has been run with `-install` previously.

- `--run-as-service`

On Unix, daemonize and start an angel process. The angel process monitors Instance Manager and restarts it if it crashes. (The angel process itself is simple and unlikely to crash.)

- `--socket=path`

On Unix, the socket file to use for incoming connections. The default file is named `/tmp/mysqlmanager.sock`. This option has no meaning on Windows.

- `--standalone`

This option is used on Windows to run Instance Manager in standalone mode. You should specify it when you start Instance Manager from the command line.

- `--user=user_name`

On Unix, the username of the system account to use for starting and running `mysqlmanager`. This option generates a warning and has no effect unless you start `mysqlmanager` as `root` (so that it can change its effective user ID), or as the named user. It is recommended that you configure `mysqlmanager` to run using the same account used to run the `mysqld` server. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

- `--username=user_name, -u user_name`

Specify the username for an entry to be added to or modified in the password file. This option was added in MySQL 5.1.12.

- `--version, -V`

Display version information and exit.

- `--wait-timeout=N`

The number of seconds to wait for activity on an incoming connection before closing it. The default is 28800 seconds (8 hours).

This option was added in MySQL 5.1.7. Before that, the timeout is 30 seconds and cannot be changed.

5.4.2. MySQL Instance Manager Configuration Files

Instance Manager uses its standard configuration file unless it is started with a `--defaults-file` option that specifies a different file. On Windows, the standard file is `my.ini` in the directory where Instance Manager is installed. On Unix, the standard file is `/etc/my.cnf`.

Instance Manager reads options for itself from the `[manager]` section of the configuration file, and options for server instances from `[mysqld]` or `[mysqldN]` sections. The `[manager]` section contains any of the options listed in [Section 5.4.1, “MySQL Instance Manager Command Options”](#), except for those specified as having to be given as the first option on the command line. Here is a sample `[manager]` section:

```
# MySQL Instance Manager options section
[manager]
default-mysqld-path = /usr/local/mysql/libexec/mysqld
socket=/tmp/manager.sock
pid-file=/tmp/manager.pid
password-file = /home/cps/.mysqlmanager.passwd
monitoring-interval = 2
port = 1999
bind-address = 192.168.1.5
```

Each `[mysqld]` or `[mysqldN]` instance section specifies options given by Instance Manager to a server instance at startup. These are mainly common MySQL Server options (see [Section 5.2.2, “Command Options”](#)). In addition, a `[mysqldN]` section can contain the options in the following list, which are specific to Instance Manager. These options are interpreted by Instance Manager itself; it does not pass them to the server when it attempts to start that server.

Warning

The Instance Manager-specific options must not be used in a `[mysqld]` section. If a server is started without using Instance Manager, it will not recognize these options and will fail to start properly.

- `mysqld-path = path`

The pathname of the `mysqld` server binary to use for the server instance.

- `nonguarded`

This option disables Instance Manager monitoring functionality for the server instance. By default, an instance is guarded: At Instance Manager start time, it starts the instance. It also monitors the instance status and attempts to restart it if it fails. At Instance Manager exit time, it stops the instance. None of these things happen for nonguarded instances.

- `shutdown-delay = seconds`

The number of seconds Instance Manager should wait for the server instance to shut down. The default value is 35 seconds. After the delay expires, Instance Manager assumes that the instance is hanging and attempts to terminate it. If you use [InnoDB](#) with large tables, you should increase this value.

Here are some sample instance sections:

```
[mysqld1]
mysqld-path=/usr/local/mysql/libexec/mysqld
socket=/tmp/mysql.sock
port=3307
server_id=1
skip-stack-trace
core-file
```

```
log-bin
log-error
log=mylog
log-slow-queries

[mysqld2]
nonguarded
port=3308
server_id=2
mysqld-path= /home/cps/mysql/trees/mysql-5.1/sql/mysqld
socket      = /tmp/mysql.sock5
pid-file    = /tmp/hostname.pid5
datadir= /home/cps/mysql_data/data_dir1
language=/home/cps/mysql/trees/mysql-5.1/sql/share/english
log-bin
log=/tmp/fordel.log
```

5.4.3. Starting the MySQL Server with MySQL Instance Manager

This section discusses how Instance Manager starts server instances when it starts. However, before you start Instance Manager, you should set up a password file for it. Otherwise, you will not be able to connect to Instance Manager to control it after it starts. For details about creating Instance Manager accounts, see [Section 5.4.4, “Instance Manager User and Password Management”](#).

On Unix, the `mysqld` MySQL database server normally is started with the `mysql.server` script, which usually resides in the `/etc/init.d/` folder. That script invokes the `mysqld_safe` script by default. However, you can use Instance Manager instead if you modify the `/etc/my.cnf` configuration file by adding `use-manager` to the `[mysql.server]` section:

```
[mysql.server]
use-manager
```

Before MySQL 5.1.12, Instance Manager always tries to start at least one server instance: When it starts, it reads its configuration file if it exists to find server instance sections and prepare a list of instances. Instance sections have names of the form `[mysqld]` or `[mysqldN]`, where `N` is an unsigned integer (for example, `[mysqld1]`, `[mysqld2]`, and so forth).

After preparing the list of instances, Instance Manager starts the guarded instances in the list. If there are no instances, Instance Manager creates an instance named `mysqld` and attempts to start it with default (compiled-in) configuration values. This means that the Instance Manager cannot find the `mysqld` program if it is not installed in the default location. ([Section 2.1.5, “Installation Layouts”](#), describes default locations for components of MySQL distributions.) If you have installed the MySQL server in a non-standard location, you should create the Instance Manager configuration file.

The startup behavior just described is similar to that of `mysqld_safe`, which always attempts to start a server. However, it lacks the flexibility required for some operations because it is not possible to run Instance Manager in such a way that it refrains from starting any server instances. For example, you cannot invoke Instance Manager for the purpose of configuring an instance without also starting it (a task that a MySQL installer application might want to perform). Consequently, MySQL 5.1.12 introduces the following changes:

- A new option, `--mysqld-safe-compatible`, may be used to cause Instance Manager to run with startup behavior similar to that used before MySQL 5.1.12: If Instance Manager finds a `[mysqld]` instance section in the configuration file, it will start it. If Instance Manager finds no `[mysqld]` section, it creates one using default configuration values, writes a `[mysqld]` section to the configuration file if it is accessible, and starts the `mysqld` instance. Instance Manager also starts any other guarded instances listed in the configuration file.
- Without `--mysqld-safe-compatible`, Instance Manager reads its configuration file if it exists and starts instances for any guarded instance sections that it finds. If there are none, it starts no instances.

Instance Manager also stops all guarded server instances when it shuts down.

The allowable options for `[mysqldN]` server instance sections are described in [Section 5.4.2, “MySQL Instance Manager Configuration Files”](#). In these sections, you can use a special `mysqld-path=path-to-mysqld-binary` option that is recognized only by Instance Manager. Use this option to let Instance Manager know where the `mysqld` binary resides. If there are multiple instances, it may also be necessary to set other options such as `datadir` and `port`, to ensure that each instance has a different data directory and TCP/IP port number. [Section 5.12, “Running Multiple MySQL Servers on the Same Machine”](#), discusses the configuration values that must differ for each instance when you run multiple instance on the same machine.

Warning

The `[mysqld]` instance section, if it exists, must not contain any Instance Manager-specific options.

The typical Unix startup/shutdown cycle for a MySQL server with the MySQL Instance Manager enabled is as follows:

1. The `/etc/init.d/mysql` script starts MySQL Instance Manager.
2. Instance Manager starts the guarded server instances and monitors them.
3. If a server instance fails, Instance Manager restarts it.
4. If Instance Manager is shut down (for example, with the `/etc/init.d/mysql stop` command), it shuts down all server instances.

5.4.4. Instance Manager User and Password Management

The Instance Manager stores its user information in a password file. On Windows, the default is `mysqlmanager.passwd` in the directory where Instance Manager is installed. On Unix, the default file is `/etc/mysqlmanager.passwd`. To specify a different location for the password file, use the `--password-file` option.

If the password file does not exist or contains no password entries, you cannot connect to the Instance Manager.

Note

Any Instance Manager process that is running to monitor server instances does not notice changes to the password file. You must stop it and restart it after making password entry changes.

Entries in the password file have the following format, where the two fields are the account username and encrypted password, separated by a colon:

```
petr:*35110DC9B4D8140F5DE667E28C72DD2597B5C848
```

Instance Manager password encryption is the same as that used by MySQL Server. It is a one-way operation; no means are provided for decrypting encrypted passwords.

Instance Manager accounts differ somewhat from MySQL Server accounts:

- MySQL Server accounts are associated with a hostname, username, and password (see [Section 5.8.1, “MySQL Usernames and Passwords”](#)).
- Instance Manager accounts are associated with a username and password only.

This means that a client can connect to Instance Manager with a given username from any host. To limit connections so that clients can connect only from the local host, start Instance Manager with the `--bind-address=127.0.0.1` option so that it listens only to the local network interface. Remote clients will not be able to connect. Local clients can connect like this:

```
shell> mysql -h 127.0.0.1 -P 2273
```

Before MySQL 5.1.12, the only option for creating password file entries is `--passwd`, which causes Instance Manager to prompt for username and password values and display the resulting entry. You can save the output in the `/etc/mysqlmanager.passwd` password file to store it. Here is an example:

```
shell> mysqlmanager --passwd >> /etc/mysqlmanager.passwd
Creating record for new user.
Enter user name: mike
Enter password: mikepass
Re-type password: mikepass
```

At the prompts, enter the username and password for the new Instance Manager user. You must enter the password twice. It does not echo to the screen, so double entry guards against entering a different password than you intend (if the two passwords do not match, no entry is generated).

The preceding command causes the following line to be added to `/etc/mysqlmanager.passwd`:

```
mike:*BBF1F551DD9DD96A01E66EC7DDC073911BAD17BA
```

Beginning with MySQL 5.1.12, the `--passwd` option is renamed to `--print-password-line` and there are several other options for managing user accounts from the command line. For example, the `--username` and `--password` options are available on the command line for specifying the username and password for an account entry. You can use them to generate an entry with no prompting like this (type the command on a single line):

```
shell> mysqlmanager --print-password-line  
--username=mike --password=mikepass >> /etc/mysqlmanager.passwd
```

If you omit the `--username` or `--password` option, Instance Manager prompts for the required value.

`--print-password-line` causes Instance Manager to send the resulting account entry to its output, which you can append to the password file. The following list describes other account-management options that cause Instance Manager to operate directly on the password file. (These options make Instance Manager scriptable for account-management purposes.) For operations on the password file to succeed, the file must exist and it must be accessible by Instance Manager. (The exception is `--clean-password-file`, which creates the file if it does not exist. Alternatively, if there is no password file, manually create it as an empty file and ensure that its ownership and access modes allow it to be read and written by Instance Manager.) The default password file is used unless you specify a `--password-file` option.

To ensure consistent treatment of the password file, it should be owned by the system account that you use for running Instance Manager to manage server instances, and you should invoke it from that account when you use it to manage accounts in the password file.

- Create a new user:

```
mysqlmanager --add-user --username=user_name [--password=password]
```

This command adds a new entry with the given username and password to the password file. The `--username` (or `-u`) option is required. `mysqlmanager` prompts for the password if it is not given on the command line with the `--password` (or `-p`) option. The command fails if the user already exists.

- Drop an existing user:

```
mysqlmanager --drop-user --username=user_name
```

This command removes the entry with the given username from the password file. The username is required. The command fails if the user does not exist.

- Change the password for an existing user:

```
mysqlmanager --edit-user --username=user_name [--password=password]
```

This command changes the given user's password in the password file. The username is required. `mysqlmanager` prompts for the password if it is not given on the command line. The command fails if the user does not exist.

- List existing users:

```
mysqlmanager --list-users
```

This command lists the usernames of the accounts in the password file.

- Check the password file:

```
mysqlmanager --check-password-file
```

This command performs a consistency and validity check of the password file. The command fails if there is something wrong with the file.

- Empty the password file:

```
mysqlmanager --clean-password-file
```

This command empties the password file, which has the effect of dropping all users listed in it. The option creates the password file if it does not exist, so it can be used to initialize a new password file to be used for other account-management operations. Take care not to use this option to reinitialize a file containing accounts that you do not want to drop.

5.4.5. MySQL Server Instance Status Monitoring

To monitor the status of each guarded server instance, the MySQL Instance Manager attempts to connect to the instance at regular intervals using the `MySQL_Instance_Manager@localhost` user account with a password of `check_connection`.

You are *not* required to create this account for MySQL Server; in fact, it is expected that it will not exist. Instance Manager can tell that a server is operational if the server accepts the connection attempt but refuses access for the account by returning a login error. However, these failed connection attempts are logged by the server to its general query log (see [Section 5.11.3](#), “The General Query Log”).

Instance Manager also attempts a connection to nonguarded server instances when you use the `SHOW INSTANCES` or `SHOW INSTANCE STATUS` command. This is the only status monitoring done for nonguarded instances.

Instance Manager knows if a server instance fails at startup because it receives a status from the attempt. For an instance that starts but later crashes, Instance Manager receives a signal because it is the parent process of the instance.

Beginning with MySQL 5.1.12, Instance Manager tracks instance states so that it can determine which commands are allowed for each instance. For example, commands that modify an instance's configuration are allowed only while the instance is offline.

Each instance is in one of the states described in the following table. Guarded instances can be in any of the states. Nonguarded instances can only be offline or online. Instance state information is displayed in the `status` column of the `SHOW INSTANCES` and `SHOW INSTANCE STATUS` commands.

State	Meaning
<code>offline</code>	The instance has not been started and is not running.
<code>starting</code>	The instance is starting (initializing). Nonguarded instances cannot be in this state. A nonguarded instance goes directly from offline to online.
<code>stopping</code>	The instance is stopping. Nonguarded instances cannot be in this state. A nonguarded instance goes directly from online to offline, or stays offline if startup fails.
<code>online</code>	The instance has started and is running.
<code>failed</code>	The instance was online but it crashed and is being restarted by Instance Manager, or else the instance failed to start at all and Instance Manager is again attempting to start it. Nonguarded instances cannot be in this state.
<code>crashed</code>	Instance Manager failed to start the instance after several attempts. (Instance Manager will try again later.) Nonguarded instances cannot be in this state.
<code>abandoned</code>	Instance Manager was not able to start the instance, has given up, and will make no further attempts until instructed otherwise. To tell Instance Manager to try again, you must first use <code>STOP INSTANCE</code> to put the instance in offline state, and then use <code>START INSTANCE</code> to start the instance. If it is necessary to make configuration changes for the instance, you must do so after putting the instance offline and before starting it. (Instance Manager accepts configuration-changing commands only for offline instances.) Nonguarded instances cannot be in this state.

5.4.6. Connecting to MySQL Instance Manager

After you set up a password file for the MySQL Instance Manager and Instance Manager is running, you can connect to it. The MySQL client-server protocol is used to communicate with the Instance Manager. For example, you can connect to it using the standard `mysql` client program:

```
shell> mysql --port=2273 --host=im.example.org --user=mysql --password
```

Instance Manager supports the version of the MySQL client-server protocol used by the client tools and libraries distributed with MySQL 4.1 or later, so other programs that use the MySQL C API also can connect to it.

5.4.7. MySQL Instance Manager Commands

After you connect to MySQL Instance Manager, you can issue commands. The following general principles apply to Instance Manager command execution:

- Commands that take an instance name fail if the name is not a valid instance name.
- Commands that take an instance name (other than `CREATE INSTANCE`) fail if the instance does not exist.
- As of MySQL 5.1.12, commands for an instance require that the instance be in an appropriate state. You cannot configure or start an instance that is not offline. You cannot start an instance that is online.
- Instance Manager maintains information about instance configuration in an internal (in-memory) cache. Initially, this information comes from the configuration file if it exists, but some commands change the configuration of an instance. Commands that modify the configuration file fail if the file does not exist or is not accessible to Instance Manager.

As of MySQL 5.1.12, configuration-changing commands modify both the in-memory cache and the server instance section recorded in the configuration file to maintain consistency between them. For this to occur, the instance must be offline and the configuration file must be accessible and not malformed. If the configuration file cannot be updated, the command fails and the cache remains unchanged.

- On Windows, the standard file is `my.ini` in the directory where Instance Manager is installed. On Unix, the standard configuration file is `/etc/my.cnf`. To specify a different configuration file, start Instance Manager with the `--defaults-file` option.
- If a `[mysqld]` instance section exists in the configuration file, it must not contain any Instance Manager-specific options (see [Section 5.4.2, “MySQL Instance Manager Configuration Files”](#)). Therefore, you must not add any of these options if you change the configuration for an instance named `mysqld`.

The following list describes the commands that Instance Manager accepts, with examples.

- `CREATE INSTANCE instance_name [option_name[=option_value], ...]`

This command configures a new instance by creating an `[instance_name]` section in the configuration file. The command fails if *instance_name* is not a valid instance name or the instance already exists.

The created section instance is empty if no options are given. Otherwise, the options are added to the section. Options should be given in the same format used when you write options in option files. (See [Section 4.3.2, “Using Option Files”](#) for a description of the allowable syntax.) If you specify multiple options, separate them by commas.

For example, to create an instance section named `[mysqld98]`, you might write something like this were you to modify the configuration file directly:

```
[mysqld98]
basedir=/var/mysql98
```

To achieve the same effect via `CREATE INSTANCE`, issue this command to Instance Manager:

```
mysql> CREATE INSTANCE mysql98 basedir="/var/mysql98";
Query OK, 0 rows affected (0.00 sec)
```

`CREATE INSTANCE` creates the instance but does not start it.

If the instance name is the (deprecated) name `mysqld`, the option list cannot include any options that are specific to Instance Manager, such as `nonguarded` (see [Section 5.4.2, “MySQL Instance Manager Configuration Files”](#)).

This command was added in MySQL 5.1.12.

- `DROP INSTANCE instance_name`

This command removes the configuration for *instance_name* from the configuration file.

```
mysql> DROP INSTANCE mysql98;
Query OK, 0 rows affected (0.00 sec)
```

The command fails if *instance_name* is not a valid instance name, the instance does not exist, or is not offline.

This command was added in MySQL 5.1.12.

- **START INSTANCE** *instance_name*

This command attempts to start an offline instance. The command is asynchronous; it does not wait for the instance to start.

```
mysql> START INSTANCE mysql4;
Query OK, 0 rows affected (0,00 sec)
```

- **STOP INSTANCE** *instance_name*

This command attempts to stop an instance. The command is synchronous; it waits for the instance to stop.

```
mysql> STOP INSTANCE mysql4;
Query OK, 0 rows affected (0,00 sec)
```

- **SHOW INSTANCES**

Shows the names and status of all loaded instances.

```
mysql> SHOW INSTANCES;
+-----+-----+
| instance_name | status |
+-----+-----+
| mysql3       | offline |
| mysql4       | online  |
| mysql2       | offline |
+-----+-----+
```

- **SHOW INSTANCE STATUS** *instance_name*

Shows status and version information for an instance.

```
mysql> SHOW INSTANCE STATUS mysql3;
+-----+-----+-----+
| instance_name | status | version |
+-----+-----+-----+
| mysql3       | online | unknown |
+-----+-----+-----+
```

- **SHOW INSTANCE OPTIONS** *instance_name*

Shows the options used by an instance.

```
mysql> SHOW INSTANCE OPTIONS mysql3;
+-----+-----+
| option_name | value |
+-----+-----+
| instance_name | mysql3 |
| mysql-path   | /home/cps/mysql/trees/mysql-4.1/sql/mysql |
| port         | 3309  |
| socket       | /tmp/mysql.sock3 |
| pid-file     | hostname.pid3 |
| datadir      | /home/cps/mysql_data/data_dir1/ |
| language     | /home/cps/mysql/trees/mysql-4.1/sql/share/english |
+-----+-----+
```

- **SHOW *instance_name* LOG FILES**

The command lists all log files used by the instance. The result set contains the path to the log file and the log file size. If no log file path is specified in the instance section of the configuration file (for example, `log=/var/mysql.log`), the Instance Manager tries to guess its placement. If Instance Manager is unable to guess the log file placement you should specify the log file location explicitly by using a log option in the appropriate instance section of the configuration file.

```
mysql> SHOW mysql LOG FILES;
+-----+-----+-----+
| Logfile | Path | Filesize |
+-----+-----+-----+
| ERROR LOG | /home/cps/var/mysql/owlet.err | 9186 |
| GENERAL LOG | /home/cps/var/mysql/owlet.log | 471503 |
| SLOW LOG | /home/cps/var/mysql/owlet-slow.log | 4463 |
+-----+-----+-----+
```


`SHOW ... LOG FILES` displays information only about log files. If a server instance uses log tables (see [Section 5.11.1](#), “Selecting General Query and Slow Query Log Output Destinations”), no information about those tables is shown.

Log options are described in [Section 5.2.2](#), “Command Options”.

- `SHOW instance_name LOG {ERROR | SLOW | GENERAL} size[,offset_from_end]`

This command retrieves a portion of the specified log file. Because most users are interested in the latest log messages, the `size` parameter defines the number of bytes to retrieve from the end of the log. To retrieve data from the middle of the log file, specify the optional `offset_from_end` parameter. The following example retrieves 21 bytes of data, starting 23 bytes before the end of the log file and ending 2 bytes before the end:

```
mysql> SHOW mysql LOG GENERAL 21, 2;
+-----+
| Log |
+-----+
| using password: YES |
+-----+
```

- `SET instance_name.option_name[=option_value]`

This command edits the specified instance's configuration section to change or add instance options. The option is added to the section if it is not already present. Otherwise, the new setting replaces the existing one.

```
mysql> SET mysql2.port=3322;
Query OK, 0 rows affected (0.00 sec)
```

As of MySQL 5.1.12, you can specify multiple options (separated by commas), and `SET` can be used only for offline instances. Each option must indicate the instance name:

```
mysql> SET mysql2.port=3322, mysql3.nonguarded;
Query OK, 0 rows affected (0.00 sec)
```

Before MySQL 5.1.12, only a single option can be specified. Also, changes made to the configuration file do not take effect until the MySQL server is restarted. In addition, these changes are not stored in the instance manager's local cache of instance settings until a `FLUSH INSTANCES` command is executed.

- `UNSET instance_name.option_name`

This command removes an option from an instance's configuration section.

```
mysql> UNSET mysql2.port;
Query OK, 0 rows affected (0.00 sec)
```

As of MySQL 5.1.12, you can specify multiple options (separated by commas), and `UNSET` can be used only for offline instances. Each option must indicate the instance name:

```
mysql> UNSET mysql2.port, mysql4.nonguarded;
Query OK, 0 rows affected (0.00 sec)
```

Before MySQL 5.1.12, only a single option can be specified. Also, changes made to the configuration file do not take effect until the MySQL server is restarted. In addition, these changes are not stored in the instance manager's local cache of instance settings until a `FLUSH INSTANCES` command is executed.

- `FLUSH INSTANCES`

As of MySQL 5.1.12, `FLUSH INSTANCES` cannot be used unless all instances are offline. The command causes Instance Manager to reread the configuration file, update its in-memory configuration cache, and start any guarded instances.

Before MySQL 5.1.12, this command forces Instance Manager reread the configuration file and to refresh internal structures. This command should be performed after editing the configuration file. The command does not restart instances.

```
mysql> FLUSH INSTANCES;
Query OK, 0 rows affected (0.04 sec)
```

`FLUSH INSTANCES` is deprecated and will be removed in MySQL 5.2.

5.5. Installation-Related Programs

The programs in this section are used when installing or upgrading MySQL.

5.5.1. `comp_err` — Compile MySQL Error Message File

`comp_err` creates the `errmsg.sys` file that is used by `mysqld` to determine the error messages to display for different error codes. `comp_err` normally is run automatically when MySQL is built. It compiles the `errmsg.sys` file from the plaintext file located at `sql/share/errmsg.txt` in MySQL source distributions.

`comp_err` also generates `mysqld_error.h`, `mysqld_ename.h`, and `sql_state.h` header files.

For more information about how error messages are defined, see the MySQL Internals Manual.

Invoke `comp_err` like this:

```
shell> comp_err [options]
```

`comp_err` understands the options described in the following list.

- `--help, -?`
Display a help message and exit.
- `--charset=path, -C path`
The character set directory. The default is `../sql/share/charsets`.
- `--debug=debug_options, -# debug_options`
Write a debugging log. The `debug_options` string often is `'d:t:0,file_name'`. The default is `'d:t:0,/tmp/comp_err.trace'`.
- `--debug-info, -T`
Print some debugging information when the program exits.
- `--header_file=file_name, -H file_name`
The name of the error header file. The default is `mysqld_error.h`.
- `--in_file=file_name, -F file_name`
The name of the input file. The default is `../sql/share/errmsg.txt`.
- `--name_file=file_name, -N file_name`
The name of the error name file. The default is `mysqld_ename.h`.
- `--out_dir=path, -D path`
The name of the output base directory. The default is `../sql/share/`.
- `--out_file=file_name, -O file_name`
The name of the output file. The default is `errmsg.sys`.
- `--statefile=file_name, -S file_name`
The name for the SQLSTATE header file. The default is `sql_state.h`.

- `--version, -V`

Display version information and exit.

5.5.2. `make_win_bin_dist` — Package MySQL Distribution as ZIP Archive

This script is used on Windows after building a MySQL distribution from source to create executable programs. It packages the binaries and support files into a ZIP archive that can be unpacked at the location where you want to install MySQL.

`make_win_bin_dist` is a shell script, so you must have Cygwin installed to use it.

This program's use is subject to change. Currently, you invoke it as follows from the root directory of your source distribution:

```
shell> make_win_bin_dist [options] package_basename [copy_def ...]
```

The `package_basename` argument provides the basename for the resulting ZIP archive. This name will be the name of the directory that results from unpacking the archive.

Because you might want to include files of directories from other builds, you can instruct this script do copy them in for you, via `copy_def` arguments, which of which is of the form `relative_dest_name=source_name`.

Example:

```
bin/mysqld-max.exe=../my-max-build/sql/release/mysqld.exe
```

If you specify a directory, the entire directory will be copied.

`make_win_bin_dist` understands the following options:

- `--debug`
Pack the debug binaries and produce an error if they were not built.
- `--embedded`
Pack the embedded server and produce an error if it was not built. The default is to pack it if it was built.
- `--exe-suffix=suffix`
Add a suffix to the basename of the `mysql` binary. For example, a suffix of `-abc` produces a binary named `mysqld-abc.exe`.
- `--no-debug`
Don't pack the debug binaries even if they were built.
- `--no-embedded`
Don't pack the embedded server even if it was built.
- `--only-debug`
Use this option when the target for this build was `Debug`, and you just want to replace the normal binaries with debug versions (that is, do not use separate `debug` directories).

5.5.3. `mysql_fix_privilege_tables` — Upgrade MySQL System Tables

Some releases of MySQL introduce changes to the structure of the system tables in the `mysql` database to add new privileges or support new features. When you update to a new version of MySQL, you should update your system tables as well to make sure that their structure is up to date. Otherwise, there might be capabilities that you cannot take advantage of. First, make a backup of your `mysql` database, and then use the following procedure.

Note

As of MySQL 5.1.7, `mysql_fix_privilege_tables` is superseded by `mysql_upgrade`, which should be used instead. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

On Unix or Unix-like systems, update the system tables by running the `mysql_fix_privilege_tables` script:

```
shell> mysql_fix_privilege_tables
```

You must run this script while the server is running. It attempts to connect to the server running on the local host as `root`. If your `root` account requires a password, indicate the password on the command line like this:

```
shell> mysql_fix_privilege_tables --password=root_password
```

The `mysql_fix_privilege_tables` script performs any actions necessary to convert your system tables to the current format. You might see some `Duplicate column name` warnings as it runs; you can ignore them.

After running the script, stop the server and restart it so that it uses any changes that were made to the system tables.

On Windows systems, MySQL distributions include a `mysql_fix_privilege_tables.sql` SQL script that you can run using the `mysql` client. For example, if your MySQL installation is located at `C:\Program Files\MySQL\MySQL Server 5.1`, the commands look like this:

```
C:\> cd "C:\Program Files\MySQL\MySQL Server 5.1"
C:\> bin\mysql -u root -p mysql
mysql> SOURCE scripts/mysql_fix_privilege_tables.sql
```

The `mysql` command will prompt you for the `root` password; enter it when prompted.

If your installation is located in some other directory, adjust the pathnames appropriately.

As with the Unix procedure, you might see some `Duplicate column name` warnings as `mysql` processes the statements in the `mysql_fix_privilege_tables.sql` script; you can ignore them.

After running the script, stop the server and restart it.

5.5.4. `mysql_install_db` — Initialize MySQL Data Directory

`mysql_install_db` initializes the MySQL data directory and creates the system tables that it contains, if they do not exist. Because the MySQL server, `mysqld`, needs to access the data directory when it runs later, you should either run `mysql_install_db` from the same account that will be used for running `mysqld` or run it as `root` and use the `--user` option to indicate the username that `mysqld` will run as.

To invoke `mysql_install_db`, use the following syntax:

```
shell> mysql_install_db [options]
```

`mysql_install_db` needs to invoke `mysqld` with the `--bootstrap` and `--skip-grant-tables` options (see [Section 2.9.2, “Typical configure Options”](#)). If MySQL was configured with the `--disable-grant-options` option, `--bootstrap` and `--skip-grant-tables` will be disabled. To handle this, set the `MYSQLD_BOOTSTRAP` environment variable to the full pathname of a server that has all options enabled. `mysql_install_db` will use that server.

`mysql_install_db` supports the following options:

- `--basedir=path`

The path to the MySQL installation directory.

- `--force`

Causes `mysql_install_db` to run even if DNS does not work. In that case, grant table entries that normally use hostnames will use IP addresses.

- `--datadir=path, --ldata=path`

The path to the MySQL data directory.

- `--rpm`

For internal use. This option is used by RPM files during the MySQL installation process.

- `--skip-name-resolve`

Use IP addresses rather than hostnames when creating grant table entries. This option can be useful if your DNS does not work.

- `--srcdir=path`

For internal use. The directory under which `mysql_install_db` looks for support files such as the error message file and the file for populating the help tables. This option was added in MySQL 5.1.14.

- `--user=user_name`

The login username to use for running `mysqld`. Files and directories created by `mysqld` will be owned by this user. You must be `root` to use this option. By default, `mysqld` runs using your current login name and files and directories that it creates will be owned by you.

- `--verbose`

Verbose mode. Print more information about what the program does.

- `--windows`

For internal use. This option is used for creating Windows distributions.

5.5.5. `mysql_secure_installation` — Improve MySQL Installation Security

This program enables you to improve the security of your MySQL installation in the following ways:

- You can set a password for `root` accounts.
- You can remove `root` accounts that are accessible from outside the local host.
- You can remove anonymous-user accounts.
- You can remove the `test` database, which by default can be accessed by anonymous users.

Invoke `mysql_secure_installation` without arguments:

```
shell> mysql_secure_installation
```

The script will prompt you to determine which actions to perform.

5.5.6. `mysql_tzinfo_to_sql` — Load the Time Zone Tables

The `mysql_tzinfo_to_sql` program loads the time zone tables in the `mysql` database. It is used on systems that have a `zoneinfo` database (the set of files describing time zones). Examples of such systems are Linux, FreeBSD, Sun Solaris, and Mac OS X. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system does not have a `zoneinfo` database, you can use the downloadable package described in [Section 5.10.8, “MySQL Server Time Zone Support”](#).

`mysql_tzinfo_to_sql` can be invoked several ways:

```
shell> mysql_tzinfo_to_sql tz_dir
shell> mysql_tzinfo_to_sql tz_file tz_name
shell> mysql_tzinfo_to_sql --leap tz_file
```

For the first invocation syntax, pass the zoneinfo directory pathname to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

The second syntax causes `mysql_tzinfo_to_sql` to load a single time zone file `tz_file` that corresponds to a time zone name `tz_name`:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

If your time zone needs to account for leap seconds, invoke `mysql_tzinfo_to_sql` using the third syntax, which initializes the leap second information. `tz_file` is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

5.5.7. `mysql_upgrade` — Check Tables for MySQL Upgrade

`mysql_upgrade` should be executed each time you upgrade MySQL. It checks all tables in all databases for incompatibilities with the current version of MySQL Server. If a table is found to have a possible incompatibility, it is checked. If any problems are found, the table is repaired. `mysql_upgrade` also upgrades the system tables so that you can take advantage of new privileges or capabilities that might have been added.

All checked and repaired tables are marked with the current MySQL version number. This ensures that next time you run `mysql_upgrade` with the same version of the server, it can tell whether there is any need to check or repair the table again.

`mysql_upgrade` also saves the MySQL version number in a file named `mysql_upgrade_info` in the data directory. This is used to quickly check if all tables have been checked for this release so that table-checking can be skipped. To ignore this file, use the `-force` option.

To check and repair tables and to upgrade the system tables, `mysql_upgrade` executes the following commands:

```
mysqlcheck --check-upgrade --all-databases --auto-repair
mysql_fix_privilege_tables
```

`mysql_upgrade` supersedes the older `mysql_fix_privilege_tables` script. In MySQL 5.1.7, `mysql_upgrade` was added as a shell script and worked only for Unix systems. As of MySQL 5.1.10, `mysql_upgrade` is an executable binary and is available on all systems. On systems older than those supporting `mysql_upgrade`, you can execute the `mysqlcheck` command manually, and then upgrade your system tables as described in [Section 5.5.3, “mysql_fix_privilege_tables — Upgrade MySQL System Tables”](#).

For details about what is checked, see the description of the `FOR UPGRADE` option of the `CHECK TABLE` statement (see [Section 12.5.2.3, “CHECK TABLE Syntax”](#)).

To use `mysql_upgrade`, make sure that the server is running, and then invoke it like this:

```
shell> mysql_upgrade [options]
```

After running `mysql_upgrade`, stop the server and restart it so that it uses any changes that were made to the system tables.

`mysql_upgrade` reads options from the command line and from the `[mysql_upgrade]` group in option files. It supports the following options:

- `--help`

Display a short help message and exit.

- `--basedir=path`

The path to the MySQL installation directory.

- `--datadir=path`

The path to the data directory.

- `--debug-check`

Print some debugging information when the program exits. This option was added in MySQL 5.1.21.

- `--debug-info, -T`

Print debugging information and memory and CPU usage statistics when the program exits. This option was added in MySQL 5.1.21.

- `--force`

Force execution of `mysqlcheck` even if `mysql_upgrade` has already been executed for the current version of MySQL. (In other words, this option causes the `mysql_upgrade_info` file to be ignored.)

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server. The default username is `root`.

- `--verbose`

Verbose mode. Print more information about what the program does.

Other options are passed to `mysqlcheck` and to `mysql_fix_privilege_tables`. For example, it might be necessary to specify the `--password[=password]` option.

5.6. General Security Issues

This section describes some general security issues to be aware of and what you can do to make your MySQL installation more secure against attack or misuse. For information specifically about the access control system that MySQL uses for setting up user accounts and checking database access, see [Section 5.7, “The MySQL Access Privilege System”](#).

For answers to some questions that are often asked about MySQL Server security issues, see [Section A.9, “MySQL 5.1 FAQ — Security”](#).

5.6.1. General Security Guidelines

Anyone using MySQL on a computer connected to the Internet should read this section to avoid the most common security mistakes.

In discussing security, we emphasize the necessity of fully protecting the entire server host (not just the MySQL server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. We do not cover all aspects of availability and fault tolerance here.

MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that users can attempt to perform. There is also support for SSL-encrypted connections between MySQL clients and servers. Many of the concepts discussed here are not specific to MySQL at all; the same general ideas apply to almost all applications.

When running MySQL, follow these guidelines whenever possible:

- **Do not ever give anyone (except MySQL `root` accounts) access to the `user` table in the `mysql` database!** This is critical.
- Learn the MySQL access privilege system. The `GRANT` and `REVOKE` statements are used for controlling access to MySQL. Do not grant more privileges than necessary. Never grant privileges to all hosts.

Checklist:

- Try `mysql -u root`. If you are able to connect successfully to the server without being asked for a password, anyone can connect to your MySQL server as the MySQL `root` user with full privileges! Review the MySQL installation instructions, paying particular attention to the information about setting a `root` password. See [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

- Use the `SHOW GRANTS` statement to check which accounts have access to what. Then use the `REVOKE` statement to remove those privileges that are not necessary.
- Do not store any plain-text passwords in your database. If your computer becomes compromised, the intruder can take the full list of passwords and use them. Instead, use `MD5()`, `SHA1()`, or some other one-way hashing function and store the hash value.
- Do not choose passwords from dictionaries. Special programs exist to break passwords. Even passwords like “xfish98” are very bad. Much better is “duag98” which contains the same word “fish” but typed one key to the left on a standard QWERTY keyboard. Another method is to use a password that is taken from the first characters of each word in a sentence (for example, “Mary had a little lamb” results in a password of “Mhall”). The password is easy to remember and type, but difficult to guess for someone who does not know the sentence.
- Invest in a firewall. This protects you from at least 50% of all types of exploits in any software. Put MySQL behind the firewall or in a demilitarized zone (DMZ).

Checklist:

- Try to scan your ports from the Internet using a tool such as `nmap`. MySQL uses port 3306 by default. This port should not be accessible from untrusted hosts. Another simple way to check whether or not your MySQL port is open is to try the following command from some remote machine, where `server_host` is the hostname or IP number of the host on which your MySQL server runs:

```
shell> telnet server_host 3306
```

If you get a connection and some garbage characters, the port is open, and should be closed on your firewall or router, unless you really have a good reason to keep it open. If `telnet` hangs or the connection is refused, the port is blocked, which is how you want it to be.

- Do not trust any data entered by users of your applications. They can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like “`; DROP DATABASE mysql;`”. This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques, if you do not prepare for them.

A common mistake is to protect only string data values. Remember to check numeric data as well. If an application generates a query such as `SELECT * FROM table WHERE ID=234` when a user enters the value `234`, the user can enter the value `234 OR 1=1` to cause the application to generate the query `SELECT * FROM table WHERE ID=234 OR 1=1`. As a result, the server retrieves every row in the table. This exposes every row and causes excessive server load. The simplest way to protect from this type of attack is to use single quotes around the numeric constants: `SELECT * FROM table WHERE ID='234'`. If the user enters extra information, it all becomes part of the string. In a numeric context, MySQL automatically converts this string to a number and strips any trailing non-numeric characters from it.

Sometimes people think that if a database contains only publicly available data, it need not be protected. This is incorrect. Even if it is allowable to display any row in the database, you should still protect against denial of service attacks (for example, those that are based on the technique in the preceding paragraph that causes the server to waste resources). Otherwise, your server becomes unresponsive to legitimate users.

Checklist:

- Try to enter single and double quote marks (‘`'`’ and ‘`"`’) in all of your Web forms. If you get any kind of MySQL error, investigate the problem right away.
- Try to modify dynamic URLs by adding `%22` (‘`"`’), `%23` (‘`#`’), and `%27` (‘`'`’) to them.
- Try to modify data types in dynamic URLs from numeric to character types using the characters shown in the previous examples. Your application should be safe against these and similar attacks.
- Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous!
- Check the size of data before passing it to MySQL.
- Have your application connect to the database using a username different from the one you use for administrative purposes. Do not give your applications any access privileges they do not need.

- Many application programming interfaces provide a means of escaping special characters in data values. Properly used, this prevents application users from entering values that cause the application to generate statements that have a different effect than you intend:
 - MySQL C API: Use the `mysql_real_escape_string()` API call.
 - MySQL++: Use the `escape` and `quote` modifiers for query streams.
 - PHP: Use the `mysql_real_escape_string()` function (available as of PHP 4.3.0, prior to that PHP version use `mysql_escape_string()`, and prior to PHP 4.0.3, use `addslashes()`). Note that only `mysql_real_escape_string()` is character set-aware; the other functions can be “bypassed” when using (invalid) multi-byte character sets. In PHP 5, you can use the `mysqli` extension, which supports the improved MySQL authentication protocol and passwords, as well as prepared statements with placeholders.
 - Perl DBI: Use placeholders or the `quote()` method.
 - Ruby DBI: Use placeholders or the `quote()` method.
 - Java JDBC: Use a `PreparedStatement` object and placeholders.

Other programming interfaces might have similar capabilities.

- Do not transmit plain (unencrypted) data over the Internet. This information is accessible to everyone who has the time and ability to intercept it and use it for their own purposes. Instead, use an encrypted protocol such as SSL or SSH. MySQL supports internal SSL connections as of version 4.0. Another technique is to use SSH port-forwarding to create an encrypted (and compressed) tunnel for the communication.
- Learn to use the `tcpdump` and `strings` utilities. In most cases, you can check whether MySQL data streams are unencrypted by issuing a command like the following:

```
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
```

(This works under Linux and should work with small modifications under other systems.) Warning: If you do not see plaintext data, this doesn't always mean that the information actually is encrypted. If you need high security, you should consult with a security expert.

5.6.2. Making MySQL Secure Against Attackers

When you connect to a MySQL server, you should use a password. The password is not transmitted in clear text over the connection. Password handling during the client connection sequence was upgraded in MySQL 4.1.1 to be very secure. If you are still using pre-4.1.1-style passwords, the encryption algorithm is not as strong as the newer algorithm. With some effort, a clever attacker who can sniff the traffic between the client and the server can crack the password. (See [Section 5.7.9, “Password Hashing as of MySQL 4.1”](#), for a discussion of the different password handling methods.)

MySQL Enterprise

The MySQL Enterprise Monitor enforces best practices for maximizing the security of your servers. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

All other information is transferred as text, and can be read by anyone who is able to watch the connection. If the connection between the client and the server goes through an untrusted network, and you are concerned about this, you can use the compressed protocol to make traffic much more difficult to decipher. You can also use MySQL's internal SSL support to make the connection even more secure. See [Section 5.8.7, “Using Secure Connections”](#). Alternatively, use SSH to get an encrypted TCP/IP connection between a MySQL server and a MySQL client. You can find an Open Source SSH client at <http://www.openssh.org/>, and a commercial SSH client at <http://www.ssh.com/>.

To make a MySQL system secure, you should strongly consider the following suggestions:

- Require all MySQL accounts to have a password. A client program does not necessarily know the identity of the person running it. It is common for client/server applications that the user can specify any username to the client program. For example, anyone can use the `mysql` program to connect as any other person simply by invoking it as `mysql -u other_user db_name` if `other_user` has no password. If all accounts have a password, connecting using another user's account becomes much more difficult.

For a discussion of methods for setting passwords, see [Section 5.8.5, “Assigning Account Passwords”](#).

- Never run the MySQL server as the Unix `root` user. This is extremely dangerous, because any user with the `FILE` privilege is able to cause the server to create files as `root` (for example, `~root/.bashrc`). To prevent this, `mysqld` refuses to run as `root` unless that is specified explicitly using the `--user=root` option.

`mysqld` can (and should) be run as an ordinary, unprivileged user instead. You can create a separate Unix account named `mysql` to make everything even more secure. Use this account only for administering MySQL. To start `mysqld` as a different Unix user, add a `user` option that specifies the username in the `[mysqld]` group of the `my.cnf` option file where you specify server options. For example:

```
[mysqld]
user=mysql
```

This causes the server to start as the designated user whether you start it manually or by using `mysqld_safe` or `mysql.server`. For more details, see [Section 5.6.5, “How to Run MySQL as a Normal User”](#).

Running `mysqld` as a Unix user other than `root` does not mean that you need to change the `root` username in the `user` table. *Usernames for MySQL accounts have nothing to do with usernames for Unix accounts.*

- Do not allow the use of symlinks to tables. (This capability can be disabled with the `--skip-symbolic-links` option.) This is especially important if you run `mysqld` as `root`, because anyone that has write access to the server's data directory then could delete any file in the system! See [Section 6.6.1.2, “Using Symbolic Links for Tables on Unix”](#).
- Make sure that the only Unix user with read or write privileges in the database directories is the user that `mysqld` runs as.
- Do not grant the `PROCESS` or `SUPER` privilege to non-administrative users. The output of `mysqladmin processlist` and `SHOW PROCESSLIST` shows the text of any statements currently being executed, so any user who is allowed to see the server process list might be able to see statements issued by other users such as `UPDATE user SET password=PASSWORD('not_secure')`.

`mysqld` reserves an extra connection for users who have the `SUPER` privilege, so that a MySQL `root` user can log in and check server activity even if all normal connections are in use.

The `SUPER` privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

- Do not grant the `FILE` privilege to non-administrative users. Any user that has this privilege can write a file anywhere in the filesystem with the privileges of the `mysqld` daemon. To make this a bit safer, files generated with `SELECT ... INTO OUTFILE` do not overwrite existing files and are writable by everyone.

The `FILE` privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file into a database table. This could be abused, for example, by using `LOAD DATA` to load `/etc/passwd` into a table, which then can be displayed with `SELECT`.

- If you do not trust your DNS, you should use IP numbers rather than hostnames in the grant tables. In any case, you should be very careful about creating grant table entries using hostname values that contain wildcards.
- If you want to restrict the number of connections allowed to a single account, you can do so by setting the `max_user_connections` variable in `mysqld`. The `GRANT` statement also supports resource control options for limiting the extent of server use allowed to an account. See [Section 12.5.1.3, “GRANT Syntax”](#).

5.6.3. Security-Related `mysqld` Options

The following `mysqld` options affect security:

- `--allow-suspicious-udfs`

This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. See [Section 28.3.4.6, “User-Defined Function Security Precautions”](#).

- `--local-infile[={0|1}]`

If you start the server with `--local-infile=0`, clients cannot use `LOCAL` in `LOAD DATA` statements. See [Section 5.6.4, “Security Issues with LOAD DATA LOCAL”](#).

- `--old-passwords`

Force the server to generate short (pre-4.1) password hashes for new passwords. This is useful for compatibility when the server must support older client programs. See [Section 5.7.9, “Password Hashing as of MySQL 4.1”](#).

MySQL Enterprise

The MySQL Enterprise Monitor offers advice on the security implications of using this option. For subscription information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `--safe-show-database` (*OBSOLETE*)

In previous versions of MySQL, this option caused the `SHOW DATABASES` statement to display the names of only those databases for which the user had some kind of privilege. In MySQL 5.1, this option is no longer available as this is now the default behavior, and there is a `SHOW DATABASES` privilege that can be used to control access to database names on a per-account basis. See [Section 12.5.1.3, “GRANT Syntax”](#).

- `--safe-user-create`

If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement unless the user has the `INSERT` privilege for the `mysql.user` table or any column in the table. If you want a user to have the ability to create new users that have those privileges that the user has the right to grant, you should grant the user the following privilege:

```
GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

This ensures that the user cannot change any privilege columns directly, but has to use the `GRANT` statement to give privileges to other users.

- `--secure-auth`

Disallow authentication for accounts that have old (pre-4.1) passwords.

The `mysql` client also has a `--secure-auth` option, which prevents connections to a server if the server requires a password in old format for the client account.

- `--secure-file-priv=path`

This option limits the effect of the `LOAD_FILE()` function and the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements to work only with files in the specified directory.

This option was added in MySQL 5.1.17.

- `--skip-grant-tables`

This option causes the server not to use the privilege system at all. This gives anyone with access to the server *unrestricted access to all databases*. You can cause a running server to start using the grant tables again by executing `mysqladmin flush-privileges` or `mysqladmin reload` command from a system shell, or by issuing a MySQL `FLUSH PRIVILEGES` statement. This option also suppresses loading of plugins and user-defined functions (UDFs).

- `--skip-merge`

Disable the `MERGE` storage engine. This option was added in MySQL 5.1.12. It can be used if the following behavior is undesirable: If a user has access to `MyISAM` table `t`, that user can create a `MERGE` table `m` that accesses `t`. However, if the user's privileges on `t` are subsequently revoked, the user can continue to access `t` by doing so through `m`.

- `--skip-name-resolve`

Hostnames are not resolved. All `Host` column values in the grant tables must be IP numbers or `localhost`.

- `--skip-networking`

Do not allow TCP/IP connections over the network. All connections to `mysqld` must be made via Unix socket files.

- `--skip-show-database`

With this option, the `SHOW DATABASES` statement is allowed only to users who have the `SHOW DATABASES` privilege, and the statement displays all database names. Without this option, `SHOW DATABASES` is allowed to all users, but displays each database name only if the user has the `SHOW DATABASES` privilege or some privilege for the database. Note that any global privilege is a privilege for the database.

- `--ssl*`

Options that begin with `--ssl` specify whether to allow clients to connect via SSL and indicate where to find SSL keys and certificates. See [Section 5.8.7.3, “SSL Command Options”](#).

5.6.4. Security Issues with `LOAD DATA LOCAL`

The `LOAD DATA` statement can load a file that is located on the server host, or it can load a file that is located on the client host when the `LOCAL` keyword is specified.

There are two potential security issues with supporting the `LOCAL` version of `LOAD DATA` statements:

- The transfer of the file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the `LOAD DATA` statement. Such a server could access any file on the client host to which the client user has read access.
- In a Web environment where the clients are connecting from a Web server, a user could use `LOAD DATA LOCAL` to read any files that the Web server process has read access to (assuming that a user could run any command against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not the remote program being run by the user who connects to the Web server.

To deal with these problems, we changed how `LOAD DATA LOCAL` is handled as of MySQL 3.23.49 and MySQL 4.0.2 (4.0.13 on Windows):

- By default, all MySQL clients and libraries in binary distributions are compiled with the `--enable-local-infile` option, to be compatible with MySQL 3.23.48 and before.
- If you build MySQL from source but do not invoke `configure` with the `--enable-local-infile` option, `LOAD DATA LOCAL` cannot be used by any client unless it is written explicitly to invoke `mysql_options(..., MYSQL_OPT_LOCAL_INFILE, 0)`. See [Section 25.2.3.49, “mysql_options\(\)”](#).
- You can disable all `LOAD DATA LOCAL` commands from the server side by starting `mysqld` with the `--local-infile=0` option.
- For the `mysql` command-line client, `LOAD DATA LOCAL` can be enabled by specifying the `--local-infile[=1]` option, or disabled with the `--local-infile=0` option. Similarly, for `mysqlimport`, the `--local` or `-L` option enables local data file loading. In any case, successful use of a local loading operation requires that the server is enabled to allow it.
- If you use `LOAD DATA LOCAL` in Perl scripts or other programs that read the `[client]` group from option files, you can add the `local-infile=1` option to that group. However, to keep this from causing problems for programs that do not understand `local-infile`, specify it using the `loose-` prefix:

```
[client]
loose-local-infile=1
```

- If `LOAD DATA LOCAL INFILE` is disabled, either in the server or the client, a client that attempts to issue such a statement receives the following error message:

```
ERROR 1148: The used command is not allowed with this MySQL version
```

MySQL Enterprise

Security advisors notify subscribers to the MySQL Enterprise Monitor whenever a server is started with the `--local-infile` option enabled. For more information see [ht-](#)

<http://www.mysql.com/products/enterprise/advisors.html>.

5.6.5. How to Run MySQL as a Normal User

On Windows, you can run the server as a Windows service using a normal user account.

On Unix, the MySQL server `mysqld` can be started and run by any user. However, you should avoid running the server as the Unix `root` user for security reasons. To change `mysqld` to run as a normal unprivileged Unix user `user_name`, you must do the following:

1. Stop the server if it's running (use `mysqladmin shutdown`).
2. Change the database directories and files so that `user_name` has privileges to read and write files in them (you might need to do this as the Unix `root` user):

```
shell> chown -R user_name /path/to/mysql/datadir
```

If you do not do this, the server will not be able to access databases or tables when it runs as `user_name`.

If directories or files within the MySQL data directory are symbolic links, you'll also need to follow those links and change the directories and files they point to. `chown -R` might not follow symbolic links for you.

3. Start the server as user `user_name`. If you are using MySQL 3.22 or later, another alternative is to start `mysqld` as the Unix `root` user and use the `--user=user_name` option. `mysqld` starts up, then switches to run as the Unix user `user_name` before accepting any connections.
4. To start the server as the given user automatically at system startup time, specify the username by adding a `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file or the `my.cnf` option file in the server's data directory. For example:

```
[mysqld]
user=user_name
```

If your Unix machine itself isn't secured, you should assign passwords to the MySQL `root` accounts in the grant tables. Otherwise, any user with a login account on that machine can run the `mysql` client with a `--user=root` option and perform any operation. (It is a good idea to assign passwords to MySQL accounts in any case, but especially so when other login accounts exist on the server host.) See [Section 2.10, "Post-Installation Setup and Testing"](#).

5.7. The MySQL Access Privilege System

MySQL has an advanced but non-standard security and privilege system. The following discussion describes how it works.

5.7.1. What the Privilege System Does

The primary function of the MySQL privilege system is to authenticate a user who connects from a given host and to associate that user with privileges on a database such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.

Additional functionality includes the ability to have anonymous users and to grant privileges for MySQL-specific functions such as `LOAD DATA INFILE` and administrative operations.

5.7.2. How the Privilege System Works

The MySQL privilege system ensures that all users may perform only the operations allowed to them. As a user, when you connect to a MySQL server, your identity is determined by *the host from which you connect* and *the username you specify*. When you issue requests after connecting, the system grants privileges according to your identity and *what you want to do*.

MySQL considers both your hostname and username in identifying you because there is little reason to assume that a given username belongs to the same person everywhere on the Internet. For example, the user `joe` who connects from `office.example.com` need not be the same person as the user `joe` who connects from `home.example.com`. MySQL handles this by allowing you to distinguish users on different hosts that happen to have the same name: You can grant one set of privileges for connections by `joe` from `office.example.com`, and a different set of privileges for connections by `joe` from `home.example.com`.

MySQL access control involves two stages when you run a client program that connects to the server:

- Stage 1: The server checks whether it should allow you to connect.
- Stage 2: Assuming that you can connect, the server checks each statement you issue to determine whether you have sufficient privileges to perform it. For example, if you try to select rows from a table in a database or drop a table from the database, the server verifies that you have the [SELECT](#) privilege for the table or the [DROP](#) privilege for the database.

If your privileges are changed (either by yourself or someone else) while you are connected, those changes do not necessarily take effect immediately for the next statement that you issue. See [Section 5.7.7, “When Privilege Changes Take Effect”](#), for details.

The server stores privilege information in the grant tables of the `mysql` database (that is, in the database named `mysql`). The MySQL server reads the contents of these tables into memory when it starts and re-reads them under the circumstances indicated in [Section 5.7.7, “When Privilege Changes Take Effect”](#). Access-control decisions are based on the in-memory copies of the grant tables.

Normally, you manipulate the contents of the grant tables indirectly by using statements such as [GRANT](#) and [REVOKE](#) to set up accounts and control the privileges available to each one. See [Section 12.5.1, “Account Management Statements”](#). The discussion here describes the underlying structure of the grant tables and how the server uses their contents when interacting with clients.

The server uses the `user`, `db`, and `host` tables in the `mysql` database at both stages of access control. The columns in the `user` and `db` tables are shown here. The `host` table is similar to the `db` table but has a specialized use as described in [Section 5.7.6, “Access Control, Stage 2: Request Verification”](#).

Table Name	user	db
Scope columns	Host	Host
	User	Db
	Password	User
Privilege columns	Select_priv	Select_priv
	Insert_priv	Insert_priv
	Update_priv	Update_priv
	Delete_priv	Delete_priv
	Index_priv	Index_priv
	Alter_priv	Alter_priv
	Create_priv	Create_priv
	Drop_priv	Drop_priv
	Grant_priv	Grant_priv
	Create_view_priv	Create_view_priv
	Show_view_priv	Show_view_priv
	Create_routine_priv	Create_routine_priv
	Alter_routine_priv	Alter_routine_priv
	Execute_priv	Execute_priv
	Trigger_priv	Trigger_priv
	Event_priv	Event_priv
	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv
	References_priv	References_priv
	Reload_priv	
	Shutdown_priv	
	Process_priv	
	File_priv	
	Show_db_priv	
	Super_priv	
	Repl_slave_priv	

	Repl_client_priv	
	Create_user_priv	
Security columns	ssl_type	
	ssl_cipher	
	x509_issuer	
	x509_subject	
Resource control columns	max_questions	
	max_updates	
	max_connections	
	max_user_connections	

The `Event_priv` and `Trigger_priv` columns were added in MySQL 5.1.6.

During the second stage of access control, the server performs request verification to make sure that each client has sufficient privileges for each request that it issues. In addition to the `user`, `db`, and `host` grant tables, the server may also consult the `tables_priv` and `columns_priv` tables for requests that involve tables. The `tables_priv` and `columns_priv` tables provide finer privilege control at the table and column levels. They have the following columns:

Table Name	tables_priv	columns_priv
Scope columns	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
Privilege columns	Table_priv	Column_priv
	Column_priv	
Other columns	Timestamp	Timestamp
	Grantor	

The `Timestamp` and `Grantor` columns currently are unused and are discussed no further here.

For verification of requests that involve stored routines, the server may consult the `procs_priv` table. This table has the following columns:

Table Name	procs_priv
Scope columns	Host
	Db
	User
	Routine_name
	Routine_type
Privilege columns	Proc_priv
Other columns	Timestamp
	Grantor

The `Routine_type` column is an `ENUM` column with values of `'FUNCTION'` or `'PROCEDURE'` to indicate the type of routine the row refers to. This column allows privileges to be granted separately for a function and a procedure with the same name.

The `Timestamp` and `Grantor` columns currently are unused and are discussed no further here.

Each grant table contains scope columns and privilege columns:

- Scope columns determine the scope of each row (entry) in the tables; that is, the context in which the row applies. For example, a `user` table row with `Host` and `User` values of `'thomas.loc.gov'` and `'bob'` would be used for authenticating connections made to the server from the host `thomas.loc.gov` by a client that specifies a username of `bob`. Similarly, a `db` table row with `Host`, `User`, and `Db` column values of `'thomas.loc.gov'`, `'bob'` and `'reports'` would be used when `bob` connects from the host `thomas.loc.gov` to access the `reports` database. The `tables_priv` and `columns_priv` tables contain scope columns indicating tables or table/column combinations to which each row applies. The `procs_priv` scope columns indicate the stored routine to which each row applies.
- Privilege columns indicate which privileges are granted by a table row; that is, what operations can be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. [Section 5.7.6, "Access Control, Stage 2: Request Verification"](#), describes the rules that are used to do this.

Scope columns contain strings. They are declared as shown here; the default value for each is the empty string:

Column Name	Type
<code>Host</code>	<code>CHAR(60)</code>
<code>User</code>	<code>CHAR(16)</code>
<code>Password</code>	<code>CHAR(16)</code>
<code>Db</code>	<code>CHAR(64)</code>
<code>Table_name</code>	<code>CHAR(64)</code>
<code>Column_name</code>	<code>CHAR(64)</code>
<code>Routine_name</code>	<code>CHAR(64)</code>

For access-checking purposes, comparisons of `Host` values are case-insensitive. `User`, `Password`, `Db`, and `Table_name` values are case sensitive. `Column_name` and `Routine_name` values are case insensitive.

In the `user`, `db`, and `host` tables, each privilege is listed in a separate column that is declared as `ENUM('N','Y') DEFAULT 'N'`. In other words, each privilege can be disabled or enabled, with the default being disabled.

In the `tables_priv`, `columns_priv`, and `procs_priv` tables, the privilege columns are declared as `SET` columns. Values in these columns can contain any combination of the privileges controlled by the table:

Table Name	Column Name	Possible Set Elements
<code>tables_priv</code>	<code>Table_priv</code>	<code>'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter', 'Create View', 'Show view', 'Trigger'</code>
<code>tables_priv</code>	<code>Column_priv</code>	<code>'Select', 'Insert', 'Update', 'References'</code>
<code>columns_priv</code>	<code>Column_priv</code>	<code>'Select', 'Insert', 'Update', 'References'</code>
<code>procs_priv</code>	<code>Proc_priv</code>	<code>'Execute', 'Alter Routine', 'Grant'</code>

Briefly, the server uses the grant tables in the following manner:

- The `user` table scope columns determine whether to reject or allow incoming connections. For allowed connections, any privileges granted in the `user` table indicate the user's global (superuser) privileges. Any privilege granted in this table applies to *all* databases on the server.

Note

Because any global privilege is considered a privilege for all databases, any global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`.

- The `db` table scope columns determine which users can access which databases from which hosts. The privilege columns determine which operations are allowed. A privilege granted at the database level applies to the database and to all its tables.

- The `host` table is used in conjunction with the `db` table when you want a given `db` table row to apply to several hosts. For example, if you want a user to be able to use a database from several hosts in your network, leave the `Host` value empty in the user's `db` table row, then populate the `host` table with a row for each of those hosts. This mechanism is described more detail in [Section 5.7.6, “Access Control, Stage 2: Request Verification”](#).

Note

The `host` table must be modified directly with statements such as `INSERT`, `UPDATE`, and `DELETE`. It is not affected by statements such as `GRANT` and `REVOKE` that modify the grant tables indirectly. Most MySQL installations need not use this table at all.

- The `tables_priv` and `columns_priv` tables are similar to the `db` table, but are more fine-grained: They apply at the table and column levels rather than at the database level. A privilege granted at the table level applies to the table and to all its columns. A privilege granted at the column level applies only to a specific column.
- The `procs_priv` table applies to stored routines. A privilege granted at the routine level applies only to a single routine.

Administrative privileges (such as `RELOAD` or `SHUTDOWN`) are specified only in the `user` table. The reason for this is that administrative operations are operations on the server itself and are not database-specific, so there is no reason to list these privileges in the other grant tables. In fact, to determine whether you can perform an administrative operation, the server need consult only the `user` table.

The `FILE` privilege also is specified only in the `user` table. It is not an administrative privilege as such, but your ability to read or write files on the server host is independent of the database you are accessing.

The `mysqld` server reads the contents of the grant tables into memory when it starts. You can tell it to re-read the tables by issuing a `FLUSH PRIVILEGES` statement or executing a `mysqladmin flush-privileges` or `mysqladmin reload` command. Changes to the grant tables take effect as indicated in [Section 5.7.7, “When Privilege Changes Take Effect”](#).

When you modify the contents of the grant tables, it is a good idea to make sure that your changes set up privileges the way you want. To check the privileges for a given account, use the `SHOW GRANTS` statement. (See [Section 12.5.4.17, “SHOW GRANTS Syntax”](#).) For example, to determine the privileges that are granted to an account with `Host` and `User` values of `pc84.example.com` and `bob`, issue this statement:

```
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

For additional help in diagnosing privilege-related problems, see [Section 5.7.8, “Causes of Access denied Errors”](#). For general advice on security issues, see [Section 5.6, “General Security Issues”](#).

5.7.3. Privileges Provided by MySQL

Information about account privileges is stored in the `user`, `db`, `host`, `tables_priv`, `columns_priv`, and `procs_priv` tables in the `mysql` database. The MySQL server reads the contents of these tables into memory when it starts and re-reads them under the circumstances indicated in [Section 5.7.7, “When Privilege Changes Take Effect”](#). Access-control decisions are based on the in-memory copies of the grant tables.

The names used in the `GRANT` and `REVOKE` statements to refer to privileges are shown in the following table, along with the column name associated with each privilege in the grant tables and the context in which the privilege applies. Further information about the meaning of each privilege may be found at [Section 12.5.1.3, “GRANT Syntax”](#).

Privilege	Column	Context
<code>CREATE</code>	<code>Create_priv</code>	databases, tables, or indexes
<code>DROP</code>	<code>Drop_priv</code>	databases or tables
<code>GRANT OPTION</code>	<code>Grant_priv</code>	databases, tables, or stored routines
<code>REFERENCES</code>	<code>References_priv</code>	databases or tables (unused)
<code>EVENT</code>	<code>Event_priv</code>	databases
<code>ALTER</code>	<code>Alter_priv</code>	tables
<code>DELETE</code>	<code>Delete_priv</code>	tables
<code>INDEX</code>	<code>Index_priv</code>	tables
<code>INSERT</code>	<code>Insert_priv</code>	tables

SELECT	Select_priv	tables
UPDATE	Update_priv	tables
TRIGGER	Trigger_priv	tables
CREATE VIEW	Create_view_priv	views
SHOW VIEW	Show_view_priv	views
ALTER ROUTINE	Alter_routine_priv	stored routines
CREATE ROUTINE	Create_routine_priv	stored routines
EXECUTE	Execute_priv	stored routines
FILE	File_priv	file access on server host
CREATE TEMPORARY TABLES	Create_tmp_table_priv	server administration
LOCK TABLES	Lock_tables_priv	server administration
CREATE USER	Create_user_priv	server administration
PROCESS	Process_priv	server administration
RELOAD	Reload_priv	server administration
REPLICATION CLIENT	Repl_client_priv	server administration
REPLICATION SLAVE	Repl_slave_priv	server administration
SHOW DATABASES	Show_db_priv	server administration
SHUTDOWN	Shutdown_priv	server administration
SUPER	Super_priv	server administration

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

The **EVENT** and **TRIGGER** privileges were added in MySQL 5.1.6.

To create or alter stored functions if binary logging is enabled, you may also need the **SUPER** privilege, as described in [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

The **CREATE** and **DROP** privileges allow you to create new databases and tables, or to drop (remove) existing databases and tables. Beginning with MySQL 5.1.10, the **DROP** privilege is also required in order to use the statement **ALTER TABLE ... DROP PARTITION** on a partitioned table. Beginning with MySQL 5.1.16, the **DROP** privilege is required for **TRUNCATE TABLE** (before that, **TRUNCATE TABLE** requires the **DELETE** privilege). *If you grant the **DROP** privilege for the **mysql** database to a user, that user can drop the database in which the MySQL access privileges are stored.*

The **SELECT**, **INSERT**, **UPDATE**, and **DELETE** privileges allow you to perform operations on rows in existing tables in a database. **INSERT** is also required for the **ANALYZE TABLE**, **OPTIMIZE TABLE**, and **REPAIR TABLE** table-maintenance statements.

SELECT statements require the **SELECT** privilege only if they actually retrieve rows from a table. Some **SELECT** statements do not access tables and can be executed without permission for any database. For example, you can use the **mysql** client as a simple calculator to evaluate expressions that make no reference to tables:

```
SELECT 1+1;
SELECT PI()*2;
```

The **INDEX** privilege enables you to create or drop (remove) indexes. **INDEX** applies to existing tables. If you have the **CREATE** privilege for a table, you can include index definitions in the **CREATE TABLE** statement.

The **ALTER** privilege enables you to use **ALTER TABLE** to change the structure of or rename tables.

The **CREATE ROUTINE** privilege is needed for creating stored routines (functions and procedures). **ALTER ROUTINE** privilege is needed for altering or dropping stored routines, and **EXECUTE** is needed for executing stored routines.

The **TRIGGER** privilege enables you to create and drop triggers. You must have this privilege for a table to create or drop triggers for that table. (Prior to MySQL 5.1.6, these operations required the **SUPER** privilege.)

The **EVENT** privilege enables you to set up events for the event scheduler.

The **GRANT** privilege enables you to give to other users those privileges that you yourself possess. It can be used for databases, tables, and stored routines.

The **FILE** privilege gives you permission to read and write files on the server host using the **LOAD DATA INFILE** and **SELECT ... INTO OUTFILE** statements. A user who has the **FILE** privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.) The **FILE** privilege also enables the user to create new files in any directory where the MySQL server has write access. As a security measure, the server will not overwrite existing files.

The remaining privileges are used for administrative operations. Many of them can be performed by using the **mysqladmin** program or by issuing SQL statements. The following table shows which **mysqladmin** commands each administrative privilege enables you to execute:

Privilege	Commands Permitted to Privilege Holders
RELOAD	flush-hosts , flush-logs , flush-privileges , flush-status , flush-tables , flush-threads , refresh , reload
SHUTDOWN	shutdown
PROCESS	processlist
SUPER	kill

The **reload** command tells the server to re-read the grant tables into memory. **flush-privileges** is a synonym for **reload**. The **refresh** command closes and reopens the log files and flushes all tables. The other **flush-xxx** commands perform functions similar to **refresh**, but are more specific and may be preferable in some instances. For example, if you want to flush just the log files, **flush-logs** is a better choice than **refresh**.

The **shutdown** command shuts down the server. There is no corresponding SQL statement.

The **processlist** command displays information about the threads executing within the server (that is, information about the statements being executed by clients). The **kill** command terminates server threads. You can always display or kill your own threads, but you need the **PROCESS** privilege to display threads initiated by other users and the **SUPER** privilege to kill them. See [Section 12.5.5.3, “KILL Syntax”](#).

The **CREATE TEMPORARY TABLES** privilege enables the use of the keyword **TEMPORARY** in **CREATE TABLE** statements.

The **LOCK TABLES** privilege enables the use of explicit **LOCK TABLES** statements to lock tables for which you have the **SELECT** privilege. This includes the use of write locks, which prevents anyone else from reading the locked table.

The **REPLICATION CLIENT** privilege enables the use of **SHOW MASTER STATUS** and **SHOW SLAVE STATUS**.

The **REPLICATION SLAVE** privilege should be granted to accounts that are used by slave servers to connect to the current server as their master. Without this privilege, the slave cannot request updates that have been made to databases on the master server.

The **SHOW DATABASES** privilege allows the account to see database names by issuing the **SHOW DATABASE** statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the **--skip-show-database** option. Note that *any* global privilege is a privilege for the database.

It is a good idea to grant to an account only those privileges that it needs. You should exercise particular caution in granting the **FILE** and administrative privileges:

- The **FILE** privilege can be abused to read into a database table any files that the MySQL server can read on the server host. This includes all world-readable files and files in the server's data directory. The table can then be accessed using **SELECT** to transfer its contents to the client host.
- The **GRANT** privilege enables users to give their privileges to other users. Two users that have different privileges and with the **GRANT** privilege are able to combine privileges.
- The **ALTER** privilege may be used to subvert the privilege system by renaming tables.
- The **SHUTDOWN** privilege can be abused to deny service to other users entirely by terminating the server.

- The **PROCESS** privilege can be used to view the plain text of currently executing statements, including statements that set or change passwords.
- The **SUPER** privilege can be used to terminate other clients or change how the server operates.
- Privileges granted for the **mysql** database itself can be used to change passwords and other access privilege information. Passwords are stored encrypted, so a malicious user cannot simply read them to know the plain text password. However, a user with write access to the **user** table **Password** column can change an account's password, and then connect to the MySQL server using that account.

MySQL Enterprise

Accounts with unnecessary global privileges constitute a security risk. Subscribers to the MySQL Enterprise Monitor are automatically alerted to the existence of such accounts. For detailed information see <http://www.mysql.com/products/enterprise/advisors.html>.

There are some things that you cannot do with the MySQL privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.
- A password applies globally to an account. You cannot associate a password with a specific object such as a database, table, or routine.

5.7.4. Connecting to the MySQL Server

MySQL client programs generally expect you to specify certain connection parameters when you want to access a MySQL server:

- The name of the host where the MySQL server is running
- Your username
- Your password

For example, the **mysql** client can be started as follows from a command-line prompt (indicated here by **shell>**):

```
shell> mysql -h host_name -u user_name -p your_pass
```

Alternative forms of the **-h**, **-u**, and **-p** options are **--host=host_name**, **--user=user_name**, and **--password=your_pass**. Note that there is *no space* between **-p** or **--password=** and the password following it.

If you use a **-p** or **--password** option but do not specify the password value, the client program prompts you to enter the password. The password is not displayed as you enter it. This is more secure than giving the password on the command line. Any user on your system may be able to see a password specified on the command line by executing a command such as **ps auxw**. See [Section 5.8.6, “Keeping Your Password Secure”](#).

MySQL client programs use default values for any connection parameter option that you do not specify:

- The default hostname is **localhost**.
- The default username is **ODBC** on Windows and your Unix login name on Unix.
- No password is supplied if neither **-p** nor **--password** is given.

Thus, for a Unix user with a login name of **joe**, all of the following commands are equivalent:

```
shell> mysql -h localhost -u joe
shell> mysql -h localhost
```

```
shell> mysql -u joe
shell> mysql
```

Other MySQL clients behave similarly.

You can specify different default values to be used when you make a connection so that you need not enter them on the command line each time you invoke a client program. This can be done in a couple of ways:

- You can specify connection parameters in the `[client]` section of an option file. The relevant section of the file might look like this:

```
[client]
host=host_name
user=user_name
password=your_pass
```

Section 4.3.2, “Using Option Files”, discusses option files further.

- You can specify some connection parameters using environment variables. The host can be specified for `mysql` using `MYSQL_HOST`. The MySQL username can be specified using `USER` (this is for Windows and NetWare only). The password can be specified using `MYSQL_PWD`, although this is insecure; see Section 5.8.6, “Keeping Your Password Secure”. For a list of variables, see Section 2.14, “Environment Variables”.

5.7.5. Access Control, Stage 1: Connection Verification

When you attempt to connect to a MySQL server, the server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password. If not, the server denies access to you completely. Otherwise, the server accepts the connection, and then enters Stage 2 and waits for requests.

Your identity is based on two pieces of information:

- The client host from which you connect
- Your MySQL username

Identity checking is performed using the three `user` table scope columns (`Host`, `User`, and `Password`). The server accepts the connection only if the `Host` and `User` columns in some `user` table row match the client hostname and username and the client supplies the password specified in that row.

`Host` values in the `user` table may be specified as follows:

- A `Host` value may be a hostname or an IP number, or `'localhost'` to indicate the local host.
- You can use the wildcard characters `%` and `_` in `Host` column values. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. For example, a `Host` value of `'%'` matches any hostname, whereas a value of `'%.mysql.com'` matches any host in the `mysql.com` domain.

MySQL Enterprise

An overly broad host specifier such as `%` constitutes a security risk. The MySQL Enterprise Monitor provides safeguards against this kind of vulnerability. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- For `Host` values specified as IP numbers, you can specify a netmask indicating how many address bits to use for the network number. For example:

```
GRANT ALL PRIVILEGES ON db.* TO david@'192.58.197.0/255.255.255.0';
```

This allows `david` to connect from any client host having an IP number `client_ip` for which the following condition is true:

```
client_ip & netmask = host_ip
```

That is, for the `GRANT` statement just shown:

```
client_ip & 255.255.255.0 = 192.58.197.0
```

IP numbers that satisfy this condition and can connect to the MySQL server are those in the range from `192.58.197.0` to `192.58.197.255`.

Note: The netmask can only be used to tell the server to use 8, 16, 24, or 32 bits of the address. Examples:

- `192.0.0.0/255.0.0.0`: anything on the 192 class A network
- `192.168.0.0/255.255.0.0`: anything on the 192.168 class B network
- `192.168.1.0/255.255.255.0`: anything on the 192.168.1 class C network
- `192.168.1.1`: only this specific IP

The following netmask (28 bits) will not work:

```
192.168.0.1/255.255.255.240
```

- A blank `Host` value in a `db` table row means that its privileges should be combined with those in the row in the `host` table that matches the client hostname. The privileges are combined using an AND (intersection) operation, not OR (union). [Section 5.7.6, “Access Control, Stage 2: Request Verification”](#), discusses use of the `host` table further.

A blank `Host` value in the other grant tables is the same as `'%'`.

Because you can use IP wildcard values in the `Host` column (for example, `'144.155.166.%'` to match every host on a subnet), someone could try to exploit this capability by naming a host `144.155.166.somewhere.com`. To foil such attempts, MySQL disallows matching on hostnames that start with digits and a dot. Thus, if you have a host named something like `1.2.foo.com`, its name never matches the `Host` column of the grant tables. An IP wildcard value can match only IP numbers, not hostnames.

In the `User` column, wildcard characters are not allowed, but you can specify a blank value, which matches any name. If the `user` table row that matches an incoming connection has a blank username, the user is considered to be an anonymous user with no name, not a user with the name that the client actually specified. This means that a blank username is used for all further access checking for the duration of the connection (that is, during Stage 2).

The `Password` column can be blank. This is not a wildcard and does not mean that any password matches. It means that the user must connect without specifying a password.

Non-blank `Password` values in the `user` table represent encrypted passwords. MySQL does not store passwords in plaintext form for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the `PASSWORD()` function). The encrypted password then is used during the connection process when checking whether the password is correct. (This is done without the encrypted password ever traveling over the connection.) From MySQL's point of view, the encrypted password is the *real* password, so you should never give anyone access to it. In particular, *do not give non-administrative users read access to tables in the `mysql` database*.

MySQL 5.1 employs the stronger authentication method (first implemented in MySQL 4.1) that has better password protection during the connection process than in earlier versions. It is secure even if TCP/IP packets are sniffed or the `mysql` database is captured. [Section 5.7.9, “Password Hashing as of MySQL 4.1”](#), discusses password encryption further.

The following table shows how various combinations of `Host` and `User` values in the `user` table apply to incoming connections.

Host Value	User Value	Allowable Connections
<code>'thomas.loc.gov'</code>	<code>'fred'</code>	<code>fred</code> , connecting from <code>thomas.loc.gov</code>
<code>'thomas.loc.gov'</code>	<code>''</code>	Any user, connecting from <code>thomas.loc.gov</code>
<code>'%'</code>	<code>'fred'</code>	<code>fred</code> , connecting from any host
<code>'%'</code>	<code>''</code>	Any user, connecting from any host
<code>'%.loc.gov'</code>	<code>'fred'</code>	<code>fred</code> , connecting from any host in the <code>loc.gov</code> domain
<code>'x.y.%'</code>	<code>'fred'</code>	<code>fred</code> , connecting from <code>x.y.net</code> , <code>x.y.com</code> , <code>x.y.edu</code> , and so

		on (this is probably not useful)
'144.155.166.177'	'fred'	fred, connecting from the host with IP address 144.155.166.177
'144.155.166.%'	'fred'	fred, connecting from any host in the 144.155.166 class C subnet
'144.155.166.0/255.255.255.0'	'fred'	Same as previous example

It is possible for the client hostname and username of an incoming connection to match more than one row in the `user` table. The preceding set of examples demonstrates this: Several of the entries shown match a connection from `thomas.loc.gov` by `fred`.

When multiple matches are possible, the server must determine which of them to use. It resolves this issue as follows:

- Whenever the server reads the `user` table into memory, it sorts the rows.
- When a client attempts to connect, the server looks through the rows in sorted order.
- The server uses the first row that matches the client hostname and username.

To see how this works, suppose that the `user` table looks like this:

Host	User	...
%	root	...
%	jeffrey	...
localhost	root	...
localhost		...

When the server reads the table into memory, it orders the rows with the most-specific `Host` values first. Literal hostnames and IP numbers are the most specific. The pattern `'%'` means “any host” and is least specific. Rows with the same `Host` value are ordered with the most-specific `User` values first (a blank `User` value means “any user” and is least specific). For the `user` table just shown, the result after sorting looks like this:

Host	User	...
localhost	root	...
localhost		...
%	jeffrey	...
%	root	...

When a client attempts to connect, the server looks through the sorted rows and uses the first match found. For a connection from `localhost` by `jeffrey`, two of the rows from the table match: the one with `Host` and `User` values of `'localhost'` and `''`, and the one with values of `'%'` and `'jeffrey'`. The `'localhost'` row appears first in sorted order, so that is the one the server uses.

Here is another example. Suppose that the `user` table looks like this:

Host	User	...
%	jeffrey	...
thomas.loc.gov		...

The sorted table looks like this:

Host	User	...
thomas.loc.gov		...
%	jeffrey	...

A connection by `jeffrey` from `thomas.loc.gov` is matched by the first row, whereas a connection by `jeffrey` from `white-`

`house.gov` is matched by the second.

It is a common misconception to think that, for a given username, all rows that explicitly name that user are used first when the server attempts to find a match for the connection. This is simply not true. The previous example illustrates this, where a connection from `thomas.loc.gov` by `jeffrey` is first matched not by the row containing `'jeffrey'` as the `User` column value, but by the row with no username. As a result, `jeffrey` is authenticated as an anonymous user, even though he specified a username when connecting.

If you are able to connect to the server, but your privileges are not what you expect, you probably are being authenticated as some other account. To find out what account the server used to authenticate you, use the `CURRENT_USER()` function. (See [Section 11.11.3, “Information Functions”](#).) It returns a value in `user_name@host_name` format that indicates the `User` and `Host` values from the matching `user` table row. Suppose that `jeffrey` connects and issues the following query:

```
mysql> SELECT CURRENT_USER();
+-----+
| CURRENT_USER() |
+-----+
| @localhost      |
+-----+
```

The result shown here indicates that the matching `user` table row had a blank `User` column value. In other words, the server is treating `jeffrey` as an anonymous user.

Another thing you can do to diagnose authentication problems is to print out the `user` table and sort it by hand to see where the first match is being made.

5.7.6. Access Control, Stage 2: Request Verification

After you establish a connection, the server enters Stage 2 of access control. For each request that you issue via that connection, the server determines what operation you want to perform, then checks whether you have sufficient privileges to do so. This is where the privilege columns in the grant tables come into play. These privileges can come from any of the `user`, `db`, `host`, `tables_priv`, `columns_priv`, or `procs_priv` tables. (You may find it helpful to refer to [Section 5.7.2, “How the Privilege System Works”](#), which lists the columns present in each of the grant tables.)

The `user` table grants privileges that are assigned to you on a global basis and that apply no matter what the default database is. For example, if the `user` table grants you the `DELETE` privilege, you can delete rows from any table in any database on the server host! In other words, `user` table privileges are superuser privileges. It is wise to grant privileges in the `user` table only to superusers such as database administrators. For other users, you should leave all privileges in the `user` table set to `'N'` and grant privileges at more specific levels only. You can grant privileges for particular databases, tables, columns, or routines.

The `db` and `host` tables grant database-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters `'%'` and `'_'` can be used in the `Host` and `Db` columns of either table. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. If you want to use either character literally when granting privileges, you must escape it with a backslash. For example, to include the underscore character (`'_'`) as part of a database name, specify it as `'\_'` in the `GRANT` statement.
- A `'%'` `Host` value in the `db` table means “any host.” A blank `Host` value in the `db` table means “consult the `host` table for further information” (a process that is described later in this section).
- A `'%'` or blank `Host` value in the `host` table means “any host.”
- A `'%'` or blank `Db` value in either table means “any database.”
- A blank `User` value in either table matches the anonymous user.

The server reads the `db` and `host` tables into memory and sorts them at the same time that it reads the `user` table. The server sorts the `db` table based on the `Host`, `Db`, and `User` scope columns, and sorts the `host` table based on the `Host` and `Db` scope columns. As with the `user` table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching entries, it uses the first match that it finds.

The `tables_priv`, `columns_priv`, and `procs_priv` tables grant table-specific, column-specific, and routine-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters ‘%’ and ‘_’ can be used in the `Host` column. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator.
- A ‘%’ or blank `Host` value means “any host.”
- The `Db`, `Table_name`, and `Column_name` columns cannot contain wildcards or be blank.

The server sorts the `tables_priv`, `columns_priv`, and `procs_priv` tables based on the `Host`, `Db`, and `User` columns. This is similar to `db` table sorting, but simpler because only the `Host` column can contain wildcards.

The server uses the sorted tables to verify each request that it receives. For requests that require administrative privileges such as `SHUTDOWN` or `RELOAD`, the server checks only the `user` table row because that is the only table that specifies administrative privileges. The server grants access if the row allows the requested operation and denies access otherwise. For example, if you want to execute `mysqladmin shutdown` but your `user` table row doesn’t grant the `SHUTDOWN` privilege to you, the server denies access without even checking the `db` or `host` tables. (They contain no `Shutdown_priv` column, so there is no need to do so.)

For database-related requests (`INSERT`, `UPDATE`, and so on), the server first checks the user’s global (superuser) privileges by looking in the `user` table row. If the row allows the requested operation, access is granted. If the global privileges in the `user` table are insufficient, the server determines the user’s database-specific privileges by checking the `db` and `host` tables:

1. The server looks in the `db` table for a match on the `Host`, `Db`, and `User` columns. The `Host` and `User` columns are matched to the connecting user’s hostname and MySQL username. The `Db` column is matched to the database that the user wants to access. If there is no row for the `Host` and `User`, access is denied.
2. If there is a matching `db` table row and its `Host` column is not blank, that row defines the user’s database-specific privileges.
3. If the matching `db` table row’s `Host` column is blank, it signifies that the `host` table enumerates which hosts should be allowed access to the database. In this case, a further lookup is done in the `host` table to find a match on the `Host` and `Db` columns. If no `host` table row matches, access is denied. If there is a match, the user’s database-specific privileges are computed as the intersection (not the union!) of the privileges in the `db` and `host` table entries; that is, the privileges that are ‘Y’ in both entries. (This way you can grant general privileges in the `db` table row and then selectively restrict them on a host-by-host basis using the `host` table entries.)

After determining the database-specific privileges granted by the `db` and `host` table entries, the server adds them to the global privileges granted by the `user` table. If the result allows the requested operation, access is granted. Otherwise, the server successively checks the user’s table and column privileges in the `tables_priv` and `columns_priv` tables, adds those to the user’s privileges, and allows or denies access based on the result. For stored routine operations, the server uses the `procs_priv` table rather than `tables_priv` and `columns_priv`.

Expressed in boolean terms, the preceding description of how a user’s privileges are calculated may be summarized like this:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

It may not be apparent why, if the global `user` row privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database, table, and column privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute an `INSERT INTO ... SELECT` statement, you need both the `INSERT` and the `SELECT` privileges. Your privileges might be such that the `user` table row grants one privilege and the `db` table row grants the other. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either table by itself; the privileges granted by the entries in both tables must be combined.

The `host` table is not affected by the `GRANT` or `REVOKE` statements, so it is unused in most MySQL installations. If you modify it directly, you can use it for some specialized purposes, such as to maintain a list of secure servers. For example, at TcX, the `host` table contains a list of all machines on the local network. These are granted all privileges.

You can also use the `host` table to indicate hosts that are *not* secure. Suppose that you have a machine `public.your.domain` that is located in a public area that you do not consider secure. You can allow access to all hosts on your network except that machine by using `host` table entries like this:

```
+-----+-----+
| Host           | Db | ...
```

```
+-----+-----+
| public.your.domain | % | ... (all privileges set to 'N')
| %.your.domain      | % | ... (all privileges set to 'Y')
+-----+-----+
```

Naturally, you should always test your changes to the grant tables (for example, by using [SHOW GRANTS](#)) to make sure that your access privileges are actually set up the way you think they are.

5.7.7. When Privilege Changes Take Effect

When `mysqld` starts, it reads all grant table contents into memory. The in-memory tables become effective for access control at that point.

When the server reloads the grant tables, privileges for existing client connections are affected as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect at the next [USE db_name](#) statement.

Note

Client applications may cache the database name; thus, this effect may not be visible to them without actually changing to a different database or executing a [FLUSH PRIVILEGES](#) statement.

- Changes to global privileges and passwords take effect the next time the client connects.

If you modify the grant tables indirectly using statements such as [GRANT](#), [REVOKE](#), or [SET PASSWORD](#), the server notices these changes and loads the grant tables into memory again immediately.

If you modify the grant tables directly using statements such as [INSERT](#), [UPDATE](#), or [DELETE](#), your changes have no effect on privilege checking until you either restart the server or tell it to reload the tables. To reload the grant tables manually, issue a [FLUSH PRIVILEGES](#) statement or execute a `mysqladmin flush-privileges` or `mysqladmin reload` command.

If you change the grant tables directly but forget to reload them, your changes have *no effect* until you restart the server. This may leave you wondering why your changes do not seem to make any difference!

5.7.8. Causes of Access denied Errors

If you encounter problems when you try to connect to the MySQL server, the following items describe some courses of action you can take to correct the problem.

- Make sure that the server is running. If it is not running, you cannot connect to it. For example, if you attempt to connect to the server and see a message such as one of those following, one cause might be that the server is not running:

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

It might also be that the server is running, but you are trying to connect using a TCP/IP port, named pipe, or Unix socket file different from the one on which the server is listening. To correct this when you invoke a client program, specify a `--port` option to indicate the proper port number, or a `--socket` option to indicate the proper named pipe or Unix socket file. To find out where the socket file is, you can use this command:

```
shell> netstat -ln | grep mysql
```

- The grant tables must be properly set up so that the server can use them for access control. For some distribution types (such as binary distributions on Windows, or RPM distributions on Linux), the installation process initializes the `mysql` database containing the grant tables. For distributions that do not do this, you must initialize the grant tables manually by running the `mysql_install_db` script. For details, see [Section 2.10.2, “Unix Post-Installation Procedures”](#).

One way to determine whether you need to initialize the grant tables is to look for a `mysql` directory under the data directory. (The data directory normally is named `data` or `var` and is located under your MySQL installation directory.) Make sure that you have a file named `user.MYD` in the `mysql` database directory. If you do not, execute the `mysql_install_db` script. After running this script and starting the server, test the initial privileges by executing this command:

```
shell> mysql -u root test
```

The server should let you connect without error.

- After a fresh installation, you should connect to the server and set up your users and their access permissions:

```
shell> mysql -u root mysql
```

The server should let you connect because the MySQL `root` user has no password initially. That is also a security risk, so setting the password for the `root` accounts is something you should do while you're setting up your other MySQL accounts. For instructions on setting the initial passwords, see [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

MySQL Enterprise

The MySQL Enterprise Monitor enforces security-related best practices. For example, subscribers are alerted whenever there is any account without a password. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- If you have updated an existing MySQL installation to a newer version, did you run the `mysql_upgrade` script? If not, do so. The structure of the grant tables changes occasionally when new capabilities are added, so after an upgrade you should always make sure that your tables have the current structure. For instructions, see [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).
- If a client program receives the following error message when it tries to connect, it means that the server expects passwords in a newer format than the client is capable of generating:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

For information on how to deal with this, see [Section 5.7.9, “Password Hashing as of MySQL 4.1”](#), and [Section B.1.2.3, “Client does not support authentication protocol”](#).

- If you try to connect as `root` and get the following error, it means that you do not have a row in the `user` table with a `User` column value of `'root'` and that `mysqld` cannot resolve the hostname for your client:

```
Access denied for user ''@'unknown' to database mysql
```

In this case, you must restart the server with the `--skip-grant-tables` option and edit your `/etc/hosts` file or `\windows\hosts` file to add an entry for your host.

- Remember that client programs use connection parameters specified in option files or environment variables. If a client program seems to be sending incorrect default connection parameters when you have not specified them on the command line, check your environment and any applicable option files. For example, if you get `Access denied` when you run a client without any options, make sure that you have not specified an old password in any of your option files!

You can suppress the use of option files by a client program by invoking it with the `--no-defaults` option. For example:

```
shell> mysqladmin --no-defaults -u root version
```

The option files that clients use are listed in [Section 4.3.2, “Using Option Files”](#). Environment variables are listed in [Section 2.14, “Environment Variables”](#).

- If you get the following error, it means that you are using an incorrect `root` password:

```
shell> mysqladmin -u root -pxxxx ver
Access denied for user 'root'@'localhost' (using password: YES)
```

If the preceding error occurs even when you have not specified a password, it means that you have an incorrect password listed in

some option file. Try the `--no-defaults` option as described in the previous item.

For information on changing passwords, see [Section 5.8.5, “Assigning Account Passwords”](#).

If you have lost or forgotten the `root` password, you can restart `mysqld` with `--skip-grant-tables` to change the password. See [Section B.1.4.1, “How to Reset the Root Password”](#).

- If you change a password by using `SET PASSWORD`, `INSERT`, or `UPDATE`, you must encrypt the password using the `PASSWORD()` function. If you do not use `PASSWORD()` for these statements, the password will not work. For example, the following statement sets a password, but fails to encrypt it, so the user is not able to connect afterward:

```
SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
```

Instead, set the password like this:

```
SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
```

The `PASSWORD()` function is unnecessary when you specify a password using the `GRANT` or `CREATE USER` statements, or the `mysqladmin password` command. Each of those automatically uses `PASSWORD()` to encrypt the password. See [Section 5.8.5, “Assigning Account Passwords”](#), and [Section 12.5.1.1, “CREATE USER Syntax”](#).

- `localhost` is a synonym for your local hostname, and is also the default host to which clients try to connect if you specify no host explicitly.

To avoid this problem on such systems, you can use a `--host=127.0.0.1` option to name the server host explicitly. This will make a TCP/IP connection to the local `mysqld` server. You can also use TCP/IP by specifying a `--host` option that uses the actual hostname of the local host. In this case, the hostname must be specified in a `user` table row on the server host, even though you are running the client program on the same host as the server.

- If you get an `Access denied` error when trying to connect to the database with `mysql -u user_name`, you may have a problem with the `user` table. Check this by executing `mysql -u root mysql` and issuing this SQL statement:

```
SELECT * FROM user;
```

The result should include a row with the `Host` and `User` columns matching your computer's hostname and your MySQL username.

- The `Access denied` error message tells you who you are trying to log in as, the client host from which you are trying to connect, and whether you were using a password. Normally, you should have one row in the `user` table that exactly matches the hostname and username that were given in the error message. For example, if you get an error message that contains `using password: NO`, it means that you tried to log in without a password.
- If the following error occurs when you try to connect from a host other than the one on which the MySQL server is running, it means that there is no row in the `user` table with a `Host` value that matches the client host:

```
Host ... is not allowed to connect to this MySQL server
```

You can fix this by setting up an account for the combination of client hostname and username that you are using when trying to connect.

If you do not know the IP number or hostname of the machine from which you are connecting, you should put a row with `'%'` as the `Host` column value in the `user` table. After trying to connect from the client machine, use a `SELECT USER()` query to see how you really did connect. (Then change the `'%'` in the `user` table row to the actual hostname that shows up in the log. Otherwise, your system is left insecure because it allows connections from any host for the given username.)

On Linux, another reason that this error might occur is that you are using a binary MySQL version that is compiled with a different version of the `glibc` library than the one you are using. In this case, you should either upgrade your operating system or `glibc`, or download a source distribution of MySQL version and compile it yourself. A source RPM is normally trivial to compile and install, so this is not a big problem.

- If you specify a hostname when trying to connect, but get an error message where the hostname is not shown or is an IP number, it means that the MySQL server got an error when trying to resolve the IP number of the client host to a name:

```
shell> mysqladmin -u root -pxxxx -h some_hostname ver
Access denied for user 'root'@'' (using password: YES)
```

This indicates a DNS problem. To fix it, execute `mysqladmin flush-hosts` to reset the internal DNS hostname cache. See [Section 6.5.9, “How MySQL Uses DNS”](#).

Some permanent solutions are:

- Determine what is wrong with your DNS server and fix it.
- Specify IP numbers rather than hostnames in the MySQL grant tables.
- Put an entry for the client machine name in `/etc/hosts` or `\windows\hosts`.
- Start `mysqld` with the `--skip-name-resolve` option.
- Start `mysqld` with the `--skip-host-cache` option.
- On Unix, if you are running the server and the client on the same machine, connect to `localhost`. Unix connections to `localhost` use a Unix socket file rather than TCP/IP.
- On Windows, if you are running the server and the client on the same machine and the server supports named pipe connections, connect to the hostname `.` (period). Connections to `.` use a named pipe rather than TCP/IP.
- If `mysql -u root test` works but `mysql -h your_hostname -u root test` results in `Access denied` (where `your_hostname` is the actual hostname of the local host), you may not have the correct name for your host in the `user` table. A common problem here is that the `Host` value in the `user` table row specifies an unqualified hostname, but your system's name resolution routines return a fully qualified domain name (or vice versa). For example, if you have an entry with host `'tcx'` in the `user` table, but your DNS tells MySQL that your hostname is `'tcx.subnet.se'`, the entry does not work. Try adding an entry to the `user` table that contains the IP number of your host as the `Host` column value. (Alternatively, you could add an entry to the `user` table with a `Host` value that contains a wildcard; for example, `'tcx.%'`. However, use of hostnames ending with `'%'` is *insecure* and is *not* recommended!)
- If `mysql -u user_name test` works but `mysql -u user_name other_db_name` does not, you have not granted database access for `other_db_name` to the given user.
- If `mysql -u user_name` works when executed on the server host, but `mysql -h host_name -u user_name` does not work when executed on a remote client host, you have not enabled access to the server for the given username from the remote host.
- If you cannot figure out why you get `Access denied`, remove from the `user` table all entries that have `Host` values containing wildcards (entries that contain `'%'` or `'_'`). A very common error is to insert a new entry with `Host='%'` and `User='some_user'`, thinking that this allows you to specify `localhost` to connect from the same machine. The reason that this does not work is that the default privileges include an entry with `Host='localhost'` and `User=''`. Because that entry has a `Host` value `'localhost'` that is more specific than `'%'`, it is used in preference to the new entry when connecting from `localhost`! The correct procedure is to insert a second entry with `Host='localhost'` and `User='some_user'`, or to delete the entry with `Host='localhost'` and `User=''`. After deleting the entry, remember to issue a `FLUSH PRIVILEGES` statement to reload the grant tables.
- If you get the following error, you may have a problem with the `db` or `host` table:

Access to database denied

If the entry selected from the `db` table has an empty value in the `Host` column, make sure that there are one or more corresponding entries in the `host` table specifying which hosts the `db` table entry applies to.

- If you are able to connect to the MySQL server, but get an `Access denied` message whenever you issue a `SELECT ... INTO OUTFILE` or `LOAD DATA INFILE` statement, your entry in the `user` table does not have the `FILE` privilege enabled.
- If you change the grant tables directly (for example, by using `INSERT`, `UPDATE`, or `DELETE` statements) and your changes seem to be ignored, remember that you must execute a `FLUSH PRIVILEGES` statement or a `mysqladmin flush-privileges` command to cause the server to re-read the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you change the `root` password with an `UPDATE` command, you won't need to specify the new password until after you flush the privileges, because the server won't know you've changed the password yet!
- If your privileges seem to have changed in the middle of a session, it may be that a MySQL administrator has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in [Section 5.7.7, “When](#)

Privilege Changes Take Effect”.

- If you have access problems with a Perl, PHP, Python, or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -p your_pass db_name`. If you are able to connect using the `mysql` client, the problem lies with your program, not with the access privileges. (There is no space between `-p` and the password; you can also use the `--password=your_pass` syntax to specify the password. If you use the `-p --password` option with no password value, MySQL prompts you for the password.)
- For testing, start the `mysqld` server with the `--skip-grant-tables` option. Then you can change the MySQL grant tables and use the `mysqlaccess` script to check whether your modifications have the desired effect. When you are satisfied with your changes, execute `mysqladmin flush-privileges` to tell the `mysqld` server to start using the new grant tables. (Reloading the grant tables overrides the `--skip-grant-tables` option. This enables you to tell the server to begin using the grant tables again without stopping and restarting it.)
- If everything else fails, start the `mysqld` server with a debugging option (for example, `--debug=d,general,query`). This prints host and user information about attempted connections, as well as information about each command issued. See [MySQL Internals: Porting](#).
- If you have any other problems with the MySQL grant tables and feel you must post the problem to the mailing list, always provide a dump of the MySQL grant tables. You can dump the tables with the `mysqldump mysql` command. To file a bug report, see the instructions at [Section 1.8, “How to Report Bugs or Problems”](#). In some cases, you may need to restart `mysqld` with `--skip-grant-tables` to run `mysqldump`.

5.7.9. Password Hashing as of MySQL 4.1

MySQL user accounts are listed in the `user` table of the `mysql` database. Each MySQL account is assigned a password, although what is stored in the `Password` column of the `user` table is not the plaintext version of the password, but a hash value computed from it. Password hash values are computed by the `PASSWORD()` function.

MySQL uses passwords in two phases of client/server communication:

- When a client attempts to connect to the server, there is an initial authentication step in which the client must present a password that has a hash value matching the hash value stored in the `user` table for the account that the client wants to use.
- After the client connects, it can (if it has sufficient privileges) set or change the password hashes for accounts listed in the `user` table. The client can do this by using the `PASSWORD()` function to generate a password hash, or by using the `GRANT` or `SET PASSWORD` statements.

In other words, the server *uses* hash values during authentication when a client first attempts to connect. The server *generates* hash values if a connected client invokes the `PASSWORD()` function or uses a `GRANT` or `SET PASSWORD` statement to set or change a password.

The password hashing mechanism was updated in MySQL 4.1 to provide better security and to reduce the risk of passwords being intercepted. However, this new mechanism is understood only by MySQL 4.1 (and newer) servers and clients, which can result in some compatibility problems. A 4.1 or newer client can connect to a pre-4.1 server, because the client understands both the old and new password hashing mechanisms. However, a pre-4.1 client that attempts to connect to a 4.1 or newer server may run into difficulties. For example, a 3.23 `mysql` client that attempts to connect to a 5.1 server may fail with the following error message:

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

Another common example of this phenomenon occurs for attempts to use the older PHP `mysql` extension after upgrading to MySQL 4.1 or newer. (See [Section 25.3.1, “Common Problems with MySQL and PHP”](#).)

The following discussion describes the differences between the old and new password mechanisms, and what you should do if you upgrade your server but need to maintain backward compatibility with pre-4.1 clients. Additional information can be found in [Section B.1.2.3, “Client does not support authentication protocol”](#). This information is of particular importance to PHP programmers migrating MySQL databases from version 4.0 or lower to version 4.1 or higher.

■ Note

This discussion contrasts 4.1 behavior with pre-4.1 behavior, but the 4.1 behavior described here actually begins with 4.1.1. MySQL 4.1.0 is an “odd” release because it has a slightly different mechanism than that implemented in 4.1.1 and up. Differences between 4.1.0 and more recent versions are described further in MySQL 5.0 Reference Manual.

Prior to MySQL 4.1, password hashes computed by the `PASSWORD()` function are 16 bytes long. Such hashes look like this:

```
mysql> SELECT PASSWORD('mypass');
+-----+
| PASSWORD('mypass') |
+-----+
| 6f8c114b58f2ce9e   |
+-----+
```

The `Password` column of the `user` table (in which these hashes are stored) also is 16 bytes long before MySQL 4.1.

As of MySQL 4.1, the `PASSWORD()` function has been modified to produce a longer 41-byte hash value:

```
mysql> SELECT PASSWORD('mypass');
+-----+
| PASSWORD('mypass') |
+-----+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-----+
```

Accordingly, the `Password` column in the `user` table also must be 41 bytes long to store these values:

- If you perform a new installation of MySQL 5.1, the `Password` column is made 41 bytes long automatically.
- Upgrading from MySQL 4.1 (4.1.1 or later in the 4.1 series) to MySQL 5.1 should not give rise to any issues in this regard because both versions use the same password hashing mechanism. If you wish to upgrade an older release of MySQL to version 5.1, you should upgrade to version 4.1 first, then upgrade the 4.1 installation to 5.1.

A widened `Password` column can store password hashes in both the old and new formats. The format of any given password hash value can be determined two ways:

- The obvious difference is the length (16 bytes versus 41 bytes).
- A second difference is that password hashes in the new format always begin with a ‘*’ character, whereas passwords in the old format never do.

The longer password hash format has better cryptographic properties, and client authentication based on long hashes is more secure than that based on the older short hashes.

The differences between short and long password hashes are relevant both for how the server uses passwords during authentication and for how it generates password hashes for connected clients that perform password-changing operations.

The way in which the server uses password hashes during authentication is affected by the width of the `Password` column:

- If the column is short, only short-hash authentication is used.
- If the column is long, it can hold either short or long hashes, and the server can use either format:
 - Pre-4.1 clients can connect, although because they know only about the old hashing mechanism, they can authenticate only using accounts that have short hashes.
 - 4.1 and later clients can authenticate using accounts that have short or long hashes.

Even for short-hash accounts, the authentication process is actually a bit more secure for 4.1 and later clients than for older clients. In terms of security, the gradient from least to most secure is:

- Pre-4.1 client authenticating with short password hash

- 4.1 or later client authenticating with short password hash
- 4.1 or later client authenticating with long password hash

The way in which the server generates password hashes for connected clients is affected by the width of the `Password` column and by the `--old-passwords` option. A 4.1 or later server generates long hashes only if certain conditions are met: The `Password` column must be wide enough to hold long values and the `--old-passwords` option must not be given. These conditions apply as follows:

- The `Password` column must be wide enough to hold long hashes (41 bytes). If the column has not been updated and still has the pre-4.1 width of 16 bytes, the server notices that long hashes cannot fit into it and generates only short hashes when a client performs password-changing operations using `PASSWORD()`, `GRANT`, or `SET PASSWORD`. This is the behavior that occurs if you have upgraded to 4.1 but have not yet run the `mysql_fix_privilege_tables` script to widen the `Password` column.
- If the `Password` column is wide, it can store either short or long password hashes. In this case, `PASSWORD()`, `GRANT`, and `SET PASSWORD` generate long hashes unless the server was started with the `--old-passwords` option. That option forces the server to generate short password hashes instead.

The purpose of the `--old-passwords` option is to enable you to maintain backward compatibility with pre-4.1 clients under circumstances where the server would otherwise generate long password hashes. The option doesn't affect authentication (4.1 and later clients can still use accounts that have long password hashes), but it does prevent creation of a long password hash in the `user` table as the result of a password-changing operation. Were that to occur, the account no longer could be used by pre-4.1 clients. Without the `--old-passwords` option, the following undesirable scenario is possible:

- An old client connects to an account that has a short password hash.
- The client changes its own password. Without `--old-passwords`, this results in the account having a long password hash.
- The next time the old client attempts to connect to the account, it cannot, because the account has a long password hash that requires the new hashing mechanism during authentication. (Once an account has a long password hash in the `user` table, only 4.1 and later clients can authenticate for it, because pre-4.1 clients do not understand long hashes.)

This scenario illustrates that, if you must support older pre-4.1 clients, it is dangerous to run a 4.1 or newer server without using the `--old-passwords` option. By running the server with `--old-passwords`, password-changing operations do not generate long password hashes and thus do not cause accounts to become inaccessible to older clients. (Those clients cannot inadvertently lock themselves out by changing their password and ending up with a long password hash.)

The downside of the `--old-passwords` option is that any passwords you create or change use short hashes, even for 4.1 clients. Thus, you lose the additional security provided by long password hashes. If you want to create an account that has a long hash (for example, for use by 4.1 clients), you must do so while running the server without `--old-passwords`.

MySQL Enterprise

Subscribers to the MySQL Enterprise Monitor are automatically alerted whenever a server is running with the `--old-passwords` option. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

The following scenarios are possible for running a 4.1 or later server:

Scenario 1: Short `Password` column in `user` table:

- Only short hashes can be stored in the `Password` column.
- The server uses only short hashes during client authentication.
- For connected clients, password hash-generating operations involving `PASSWORD()`, `GRANT`, or `SET PASSWORD` use short hashes exclusively. Any change to an account's password results in that account having a short password hash.
- The `--old-passwords` option can be used but is superfluous because with a short `Password` column, the server generates only short password hashes anyway.

Scenario 2: Long `Password` column; server not started with `--old-passwords` option:

- Short or long hashes can be stored in the `Password` column.
- 4.1 and later clients can authenticate using accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only using accounts that have short hashes.
- For connected clients, password hash-generating operations involving `PASSWORD()`, `GRANT`, or `SET PASSWORD` use long hashes exclusively. A change to an account's password results in that account having a long password hash.

As indicated earlier, a danger in this scenario is that it is possible for accounts that have a short password hash to become inaccessible to pre-4.1 clients. A change to such an account's password made via `GRANT`, `PASSWORD()`, or `SET PASSWORD` results in the account being given a long password hash. From that point on, no pre-4.1 client can authenticate to that account until the client upgrades to 4.1.

To deal with this problem, you can change a password in a special way. For example, normally you use `SET PASSWORD` as follows to change an account password:

```
SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('mypass');
```

To change the password but create a short hash, use the `OLD_PASSWORD()` function instead:

```
SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('mypass');
```

`OLD_PASSWORD()` is useful for situations in which you explicitly want to generate a short hash.

Scenario 3: Long `Password` column; 4.1 or newer server started with `--old-passwords` option:

- Short or long hashes can be stored in the `Password` column.
- 4.1 and later clients can authenticate for accounts that have short or long hashes (but note that it is possible to create long hashes only when the server is started without `--old-passwords`).
- Pre-4.1 clients can authenticate only for accounts that have short hashes.
- For connected clients, password hash-generating operations involving `PASSWORD()`, `GRANT`, or `SET PASSWORD` use short hashes exclusively. Any change to an account's password results in that account having a short password hash.

In this scenario, you cannot create accounts that have long password hashes, because the `--old-passwords` option prevents generation of long hashes. Also, if you create an account with a long hash before using the `--old-passwords` option, changing the account's password while `--old-passwords` is in effect results in the account being given a short password, causing it to lose the security benefits of a longer hash.

The disadvantages for these scenarios may be summarized as follows:

In scenario 1, you cannot take advantage of longer hashes that provide more secure authentication.

In scenario 2, accounts with short hashes become inaccessible to pre-4.1 clients if you change their passwords without explicitly using `OLD_PASSWORD()`.

In scenario 3, `--old-passwords` prevents accounts with short hashes from becoming inaccessible, but password-changing operations cause accounts with long hashes to revert to short hashes, and you cannot change them back to long hashes while `--old-passwords` is in effect.

5.7.9.1. Implications of Password Hashing Changes for Application Programs

An upgrade to MySQL version 4.1 or later can cause compatibility issues for applications that use `PASSWORD()` to generate passwords for their own purposes. Applications really should not do this, because `PASSWORD()` should be used only to manage passwords for MySQL accounts. But some applications use `PASSWORD()` for their own purposes anyway.

If you upgrade to 4.1 or later from a pre-4.1 version of MySQL and run the server under conditions where it generates long password hashes, an application using `PASSWORD()` for its own passwords breaks. The recommended course of action in such cases is to modify the application to use another function, such as `SHA1()` or `MD5()`, to produce hashed values. If that is not possible, you can use the `OLD_PASSWORD()` function, which is provided for generate short hashes in the old format. However, you should note that

`OLD_PASSWORD()` may one day no longer be supported.

If the server is running under circumstances where it generates short hashes, `OLD_PASSWORD()` is available but is equivalent to `PASSWORD()`.

PHP programmers migrating their MySQL databases from version 4.0 or lower to version 4.1 or higher should see [Section 25.3](#), “MySQL PHP API”.

5.8. MySQL User Account Management

This section describes how to set up accounts for clients of your MySQL server. It discusses the following topics:

- The meaning of account names and passwords as used in MySQL and how that compares to names and passwords used by your operating system
- How to set up new accounts and remove existing accounts
- How to change passwords
- Guidelines for using passwords securely
- How to use secure connections with SSL

5.8.1. MySQL Usernames and Passwords

A MySQL account is defined in terms of a username and the client host or hosts from which the user can connect to the server. The account also has a password. There are several distinctions between the way usernames and passwords are used by MySQL and the way they are used by your operating system:

- Usernames, as used by MySQL for authentication purposes, have nothing to do with usernames (login names) as used by Windows or Unix. On Unix, most MySQL clients by default try to log in using the current Unix username as the MySQL username, but that is for convenience only. The default can be overridden easily, because client programs allow any username to be specified with a `-u` or `--user` option. Because this means that anyone can attempt to connect to the server using any username, you cannot make a database secure in any way unless all MySQL accounts have passwords. Anyone who specifies a username for an account that has no password is able to connect successfully to the server.
- MySQL usernames can be up to 16 characters long. This limit is hard-coded in the MySQL servers and clients, and trying to circumvent it by modifying the definitions of the tables in the `mysql` database *does not work*.

Note

You should never alter any of the tables in the `mysql` database in any manner whatsoever except by means of the procedure prescribed by MySQL AB that is described in [Section 5.5.7](#), “`mysql_upgrade` — Check Tables for MySQL Upgrade”. Attempting to redefine MySQL’s system tables in any other fashion results in undefined (and unsupported!) behavior.

Operating system usernames are completely unrelated to MySQL usernames and may even be of a different maximum length. For example, Unix usernames typically are limited to eight characters.

- MySQL passwords have nothing to do with passwords for logging in to your operating system. There is no necessary connection between the password you use to log in to a Windows or Unix machine and the password you use to access the MySQL server on that machine.
- MySQL encrypts passwords using its own algorithm. This encryption is different from that used during the Unix login process. MySQL password encryption is the same as that implemented by the `PASSWORD()` SQL function. Unix password encryption is the same as that implemented by the `ENCRYPT()` SQL function. See the descriptions of the `PASSWORD()` and `ENCRYPT()` functions in [Section 11.11.2](#), “Encryption and Compression Functions”. From version 4.1 on, MySQL employs a stronger authentication method that has better password protection during the connection process than in earlier versions. It is secure even if TCP/IP packets are sniffed or the `mysql` database is captured. (In earlier versions, even though passwords are stored in encrypted form in the `user` table, knowledge of the encrypted password value could be used to connect to the MySQL server.)

When you install MySQL, the grant tables are populated with an initial set of accounts. These accounts have names and access privileges that are described in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#), which also discusses how to assign passwords to them. Thereafter, you normally set up, modify, and remove MySQL accounts using statements such as [GRANT](#) and [REVOKE](#). See [Section 12.5.1, “Account Management Statements”](#).

When you connect to a MySQL server with a command-line client, you should specify the username and password for the account that you want to use:

```
shell> mysql --user=monty --password=guess db_name
```

If you prefer short options, the command looks like this:

```
shell> mysql -u monty -pguess db_name
```

There must be *no space* between the `-p` option and the following password value. See [Section 5.7.4, “Connecting to the MySQL Server”](#).

The preceding commands include the password value on the command line, which can be a security risk. See [Section 5.8.6, “Keeping Your Password Secure”](#). To avoid this problem, specify the `--password` or `-p` option without any following password value:

```
shell> mysql --user=monty --password db_name
shell> mysql -u monty -p db_name
```

When the password option has no password value, the client program prints a prompt and waits for you to enter the password. (In these examples, `db_name` is *not* interpreted as a password because it is separated from the preceding password option by a space.)

On some systems, the library routine that MySQL uses to prompt for a password automatically limits the password to eight characters. That is a problem with the system library, not with MySQL. Internally, MySQL doesn't have any limit for the length of the password. To work around the problem, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

5.8.2. Adding New User Accounts to MySQL

You can create MySQL accounts in two ways:

- By using statements intended for creating accounts, such as [CREATE USER](#) or [GRANT](#)
- By manipulating the MySQL grant tables directly with statements such as [INSERT](#), [UPDATE](#), or [DELETE](#)

The preferred method is to use account-creation statements because they are more concise and less error-prone. [CREATE USER](#) and [GRANT](#) are described in [Section 12.5.1.1, “CREATE USER Syntax”](#), and [Section 12.5.1.3, “GRANT Syntax”](#).

Another option for creating accounts is to use one of several available third-party programs that offer capabilities for MySQL account administration. [phpMyAdmin](#) is one such program.

The following examples show how to use the `mysql` client program to set up new users. These examples assume that privileges are set up according to the defaults described in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#). This means that to make changes, you must connect to the MySQL server as the MySQL `root` user, and the `root` account must have the [INSERT](#) privilege for the `mysql` database and the [RELOAD](#) administrative privilege.

As noted in the examples where appropriate, some of the statements will fail if you have the server's SQL mode has been set to enable certain restrictions. In particular, strict mode ([STRICT_TRANS_TABLES](#), [STRICT_ALL_TABLES](#)) and [NO_AUTO_CREATE_USER](#) will prevent the server from accepting some of the statements. Workarounds are indicated for these cases. For more information about SQL modes and their effect on grant table manipulation, see [Section 5.2.6, “SQL Modes”](#), and [Section 12.5.1.3, “GRANT Syntax”](#).

First, use the `mysql` program to connect to the server as the MySQL `root` user:

```
shell> mysql --user=root mysql
```

If you have assigned a password to the `root` account, you'll also need to supply a `--password` or `-p` option for this `mysql` command and also for those later in this section.

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost'  
-> IDENTIFIED BY 'some_pass' WITH GRANT OPTION;  
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'%'  
-> IDENTIFIED BY 'some_pass' WITH GRANT OPTION;  
mysql> GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';  
mysql> GRANT USAGE ON *.* TO 'dummy'@'localhost';
```

table privileges are global, so no entries in any of the other grant tables are needed.

The next examples create three accounts and give them access to specific databases. Each of them has a username of `custom` and password of `obscure`.

To create the accounts with `GRANT`, use the following statements:

```
shell> mysql --user=root mysql
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON bankaccount.*
-> TO 'custom'@'localhost'
-> IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON expenses.*
-> TO 'custom'@'whitehouse.gov'
-> IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON customer.*
-> TO 'custom'@'server.domain'
-> IDENTIFIED BY 'obscure';
```

The three accounts can be used as follows:

- The first account can access the `bankaccount` database, but only from the local host.
- The second account can access the `expenses` database, but only from the host `whitehouse.gov`.
- The third account can access the `customer` database, but only from the host `server.domain`.

To set up the `custom` accounts without `GRANT`, use `INSERT` statements as follows to modify the grant tables directly:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('localhost','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('whitehouse.gov','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('server.domain','custom',PASSWORD('obscure'));
mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv,Delete_priv,Create_priv,Drop_priv)
-> VALUES('localhost','bankaccount','custom',
-> 'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv,Delete_priv,Create_priv,Drop_priv)
-> VALUES('whitehouse.gov','expenses','custom',
-> 'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv,Delete_priv,Create_priv,Drop_priv)
-> VALUES('server.domain','customer','custom',
-> 'Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

The first three `INSERT` statements add `user` table entries that allow the user `custom` to connect from the various hosts with the given password, but grant no global privileges (all privileges are set to the default value of `'N'`). The next three `INSERT` statements add `db` table entries that grant privileges to `custom` for the `bankaccount`, `expenses`, and `customer` databases, but only when accessed from the proper hosts. As usual when you modify the grant tables directly, you must tell the server to reload them with `FLUSH PRIVILEGES` so that the privilege changes take effect.

If you want to give a specific user access from all machines in a given domain (for example, `mydomain.com`), you can issue a `GRANT` statement that uses the `'%'` wildcard character in the host part of the account name:

```
mysql> GRANT ...
-> ON *.*
-> TO 'myname'@'%mydomain.com'
-> IDENTIFIED BY 'mypass';
```

To do the same thing by modifying the grant tables directly, do this:

```
mysql> INSERT INTO user (Host,User,Password,...)
-> VALUES('%mydomain.com','myname',PASSWORD('mypass'),...);
mysql> FLUSH PRIVILEGES;
```

5.8.3. Removing User Accounts from MySQL

To remove an account, use the `DROP USER` statement, which is described in [Section 12.5.1.2, “DROP USER Syntax”](#).

5.8.4. Limiting Account Resources

One means of limiting use of MySQL server resources is to set the `max_user_connections` system variable to a non-zero value. However, this method is strictly global, and does not allow for management of individual accounts. In addition, it limits only the number of simultaneous connections made using a single account, and not what a client can do once connected. Both types of control are of interest to many MySQL administrators, particularly those working for Internet Service Providers.

In MySQL 5.1, you can limit the following server resources for individual accounts:

- The number of queries that an account can issue per hour
- The number of updates that an account can issue per hour
- The number of times an account can connect to the server per hour

Any statement that a client can issue counts against the query limit. Only statements that modify databases or tables count against the update limit.

It is also possible to limit the number of simultaneous connections to the server on a per-account basis.

An account in this context is a single row in the `user` table. Each account is uniquely identified by its `User` and `Host` column values.

As a prerequisite for using this feature, the `user` table in the `mysql` database must contain the resource-related columns. Resource limits are stored in the `max_questions`, `max_updates`, `max_connections`, and `max_user_connections` columns. If your `user` table doesn't have these columns, it must be upgraded; see [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

To set resource limits with a `GRANT` statement, use a `WITH` clause that names each resource to be limited and a per-hour count indicating the limit value. For example, to create a new account that can access the `customer` database, but only in a limited fashion, issue this statement:

```
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
-> IDENTIFIED BY 'frank'
-> WITH MAX_QUERIES_PER_HOUR 20
-> MAX_UPDATES_PER_HOUR 10
-> MAX_CONNECTIONS_PER_HOUR 5
-> MAX_USER_CONNECTIONS 2;
```

The limit types need not all be named in the `WITH` clause, but those named can be present in any order. The value for each per-hour limit should be an integer representing a count per hour. If the `GRANT` statement has no `WITH` clause, the limits are each set to the default value of zero (that is, no limit). For `MAX_USER_CONNECTIONS`, the limit is an integer indicating the maximum number of simultaneous connections the account can make at any one time. If the limit is set to the default value of zero, the `max_user_connections` system variable determines the number of simultaneous connections for the account.

To set or change limits for an existing account, use a `GRANT USAGE` statement at the global level (`ON *.*`). The following statement changes the query limit for `francis` to 100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
-> WITH MAX_QUERIES_PER_HOUR 100;
```

This statement leaves the account's existing privileges unchanged and modifies only the limit values specified.

To remove an existing limit, set its value to zero. For example, to remove the limit on how many times per hour `francis` can connect, use this statement:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
-> WITH MAX_CONNECTIONS_PER_HOUR 0;
```

Resource-use counting takes place when any account has a non-zero limit placed on its use of any of the resources.

As the server runs, it counts the number of times each account uses resources. If an account reaches its limit on number of connections within the last hour, further connections for the account are rejected until that hour is up. Similarly, if the account reaches its limit on the number of queries or updates, further queries or updates are rejected until the hour is up. In all such cases, an appropriate error message is issued.

Resource counting is done per account, not per client. For example, if your account has a query limit of 50, you cannot increase your limit to 100 by making two simultaneous client connections to the server. Queries issued on both connections are counted together.

Queries for which results are served from the query cache do not count against the `MAX_QUERIES_PER_HOUR` limit.

The current per-hour resource-use counts can be reset globally for all accounts, or individually for a given account:

- To reset the current counts to zero for all accounts, issue a `FLUSH USER_RESOURCES` statement. The counts also can be reset by reloading the grant tables (for example, with a `FLUSH PRIVILEGES` statement or a `mysqladmin reload` command).
- The counts for an individual account can be set to zero by re-granting it any of its limits. To do this, use `GRANT USAGE` as described earlier and specify a limit value equal to the value that the account currently has.

Counter resets do not affect the `MAX_USER_CONNECTIONS` limit.

All counts begin at zero when the server starts; counts are not carried over through a restart.

5.8.5. Assigning Account Passwords

Passwords may be assigned from the command line by using the `mysqladmin` command:

```
shell> mysqladmin -u user_name -h host_name password "newpwd"
```

The account for which this command resets the password is the one with a `user` table row that matches `user_name` in the `User` column and the client host *from which you connect* in the `Host` column.

Another way to assign a password to an account is to issue a `SET PASSWORD` statement:

```
mysql> SET PASSWORD FOR 'jeffrey'@'%' = PASSWORD('biscuit');
```

Only users such as `root` that have update access to the `mysql` database can change the password for other users. If you are not connected as an anonymous user, you can change your own password by omitting the `FOR` clause:

```
mysql> SET PASSWORD = PASSWORD('biscuit');
```

You can also use a `GRANT USAGE` statement at the global level (`ON *.*`) to assign a password to an account without affecting the account's current privileges:

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'%' IDENTIFIED BY 'biscuit';
```

Although it is generally preferable to assign passwords using one of the preceding methods, you can also do so by modifying the `user` table directly:

- To establish a password when creating a new account, provide a value for the `Password` column:

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('','%','jeffrey',PASSWORD('biscuit'));
mysql> FLUSH PRIVILEGES;
```

- To change the password for an existing account, use `UPDATE` to set the `Password` column value:

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password = PASSWORD('bagel')
-> WHERE Host = '%' AND User = 'francis';
mysql> FLUSH PRIVILEGES;
```


When you assign an account a non-empty password using `SET PASSWORD`, `INSERT`, or `UPDATE`, you must use the `PASSWORD()` function to encrypt it. `PASSWORD()` is necessary because the `user` table stores passwords in encrypted form, not as plaintext. If you forget that fact, you are likely to set passwords like this:

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('','jeffrey','biscuit');
mysql> FLUSH PRIVILEGES;
```

The result is that the literal value `'biscuit'` is stored as the password in the `user` table, not the encrypted value. When `jeffrey` attempts to connect to the server using this password, the value is encrypted and compared to the value stored in the `user` table. However, the stored value is the literal string `'biscuit'`, so the comparison fails and the server rejects the connection:

```
shell> mysql -u jeffrey -pbiscuit test
Access denied
```

If you assign passwords using the `GRANT ... IDENTIFIED BY` statement or the `mysqladmin password` command, they both take care of encrypting the password for you. In these cases, using `PASSWORD()` function is unnecessary.

Note

`PASSWORD()` encryption is different from Unix password encryption. See [Section 5.8.1, “MySQL Usernames and Passwords”](#).

5.8.6. Keeping Your Password Secure

On an administrative level, you should never grant access to the `user` grant table to any non-administrative accounts.

When you run a client program to connect to the MySQL server, it is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed here, along with an assessment of the risks of each method:

- Use a `-pyour_pass` or `--password=your_pass` option on the command line. For example:

```
shell> mysql -u francis -pfrank db_name
```

This is convenient *but insecure*, because your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. On some systems this strategy is ineffective, anyway, and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

- Use the `-p` or `--password` option with no password value specified. In this case, the client program solicits the password from the terminal:

```
shell> mysql -u francis -p db_name
Enter password: *****
```

The `*` characters indicate where you enter your password. The password is not displayed as you enter it.

It is more secure to enter your password this way than to specify it on the command line because it is not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs non-interactively, there is no opportunity to enter the password from the terminal. On some systems, you may even find that the first line of your script is read and interpreted (incorrectly) as your password.

- Store your password in an option file. For example, on Unix you can list your password in the `[client]` section of the `.my.cnf` file in your home directory:

```
[client]
password=your_pass
```

If you store your password in `.my.cnf`, the file should not be accessible to anyone but yourself. To ensure this, set the file access mode to `400` or `600`. For example:


```
shell> chmod 600 .my.cnf
```

Section 4.3.2, “Using Option Files”, discusses option files in more detail.

- Store your password in the `MYSQL_PWD` environment variable. This method of specifying your MySQL password must be considered *extremely insecure* and should not be used. Some versions of `ps` include an option to display the environment of running processes. If you set `MYSQL_PWD`, your password is exposed to any other user who runs `ps`. Even on systems without such a version of `ps`, it is unwise to assume that there are no other methods by which users can examine process environments. See Section 2.14, “Environment Variables”.

All in all, the safest methods are to have the client program prompt for the password or to specify the password in a properly protected option file.

5.8.7. Using Secure Connections

MySQL supports secure (encrypted) connections between MySQL clients and the server using the Secure Sockets Layer (SSL) protocol. This section discusses how to use SSL connections. It also describes a way to set up SSH on Windows. For information on how to require users to use SSL connections, see the discussion of the `REQUIRE` clause of the `GRANT` statement in Section 12.5.1.3, “`GRANT` Syntax”.

The standard configuration of MySQL is intended to be as fast as possible, so encrypted connections are not used by default. Doing so would make the client/server protocol much slower. Encrypting data is a CPU-intensive operation that requires the computer to do additional work and can delay other MySQL tasks. For applications that require the security provided by encrypted connections, the extra computation is warranted.

MySQL allows encryption to be enabled on a per-connection basis. You can choose a normal unencrypted connection or a secure encrypted SSL connection according the requirements of individual applications.

Secure connections are based on the OpenSSL API and are available through the MySQL C API. Replication uses the C API, so secure connections can be used between master and slave servers.

5.8.7.1. Basic SSL Concepts

To understand how MySQL uses SSL, it is necessary to explain some basic SSL and X509 concepts. People who are familiar with these can skip this part of the discussion.

By default, MySQL uses unencrypted connections between the client and the server. This means that someone with access to the network could watch all your traffic and look at the data being sent or received. They could even change the data while it is in transit between client and server. To improve security a little, you can compress client/server traffic by using the `--compress` option when invoking client programs. However, this does not foil a determined attacker.

When you need to move information over a network in a secure fashion, an unencrypted connection is unacceptable. Encryption is the way to make any kind of data unreadable. In fact, today's practice requires many additional security elements from encryption algorithms. They should resist many kind of known attacks such as changing the order of encrypted messages or replaying data twice.

SSL is a protocol that uses different encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect any data change, loss, or replay. SSL also incorporates algorithms that provide identity verification using the X509 standard.

X509 makes it possible to identify someone on the Internet. It is most commonly used in e-commerce applications. In basic terms, there should be some company called a “Certificate Authority” (or CA) that assigns electronic certificates to anyone who needs them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key and a secret key). A certificate owner can show the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted with this public key can be decrypted only using the corresponding secret key, which is held by the owner of the certificate.

If you need more information about SSL, X509, or encryption, use your favorite Internet search engine to search for the keywords in which you are interested.

5.8.7.2. Using SSL Connections

To use SSL connections between the MySQL server and client programs, your system must support either OpenSSL or yaSSL and your version of MySQL must be built with SSL support.

To make it easier to use secure connections, MySQL is bundled with yaSSL. (MySQL and yaSSL employ the same licensing model, whereas OpenSSL uses an Apache-style license.) yaSSL support initially was available only for a few platforms, but now it is available on all platforms supported by MySQL AB.

To get secure connections to work with MySQL and SSL, you must do the following:

1. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, and you are going to use OpenSSL rather than the bundled yaSSL library, install OpenSSL if it has not already been installed. We have tested MySQL with OpenSSL 0.9.6. To obtain OpenSSL, visit <http://www.openssl.org>.
2. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, configure a MySQL source distribution to use SSL. When you configure MySQL, invoke the `configure` script like this:

```
shell> ./configure --with-ssl
```

That configures the distribution to use the bundled yaSSL library. To use OpenSSL instead, specify the `--with-ssl` option with the path to the directory where the OpenSSL header files and libraries are located:

```
shell> ./configure --with-ssl=path
```

Before MySQL 5.1.11, you must use the appropriate option to select the SSL library that you want to use.

For yaSSL:

```
shell> ./configure --with-yassl
```

For OpenSSL:

```
shell> ./configure --with-openssl
```

Note that yaSSL support on Unix platforms requires that either `/dev/urandom` or `/dev/random` be available to retrieve true random numbers. For additional information (especially regarding yaSSL on Solaris versions prior to 2.8 and HP-UX), see [Bug#13164](#).

3. Make sure that you have upgraded your grant tables to include the SSL-related columns in the `mysql.user` table. This is necessary if your grant tables date from a version of MySQL older than 4.0. The upgrade procedure is described in [Section 5.5.7](#), “`mysql_upgrade` — Check Tables for MySQL Upgrade”.
4. To check whether a server binary is compiled with SSL support, invoke it with the `--ssl` option. An error will occur if the server does not support SSL:

```
shell> mysqld --ssl --help
060525 14:18:52 [ERROR] mysqld: unknown option '--ssl'
```

To check whether a running `mysqld` server supports SSL, examine the value of the `have_openssl` system variable:

```
mysql> SHOW VARIABLES LIKE 'have_openssl';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_openssl  | YES   |
+-----+-----+
```

If the value is `YES`, the server supports SSL connections. If the value is `DISABLED`, the server supports SSL connections but was not started with the appropriate `--ssl-xxx` options (described later in this section). If the value is `YES`, the server supports SSL connections.

To enable SSL connections, the proper SSL-related command options must be used (see [Section 5.8.7.3](#), “SSL Command Options”).

To start the MySQL server so that it allows clients to connect via SSL, use the options that identify the key and certificate files the server needs when establishing a secure connection:

```
shell> mysqld --ssl-ca=cacert.pem \
--ssl-cert=server-cert.pem \
```

```
--ssl-key=server-key.pem
```

- `--ssl-ca` identifies the Certificate Authority (CA) certificate.
- `--ssl-cert` identifies the server public key. This can be sent to the client and authenticated against the CA certificate that it has.
- `--ssl-key` identifies the server private key.

To establish a secure connection to a MySQL server with SSL support, the options that a client must specify depend on the SSL requirements of the user account that the client uses. (See the discussion of the `REQUIRE` clause in [Section 12.5.1.3, “GRANT Syntax”](#).)

If the account has no special SSL requirements or was created using a `GRANT` statement that includes the `REQUIRE SSL` option, a client can connect securely by using just the `--ssl-ca` option:

```
shell> mysql --ssl-ca=cacert.pem
```

To require that a client certificate also be specified, create the account using the `REQUIRE X509` option. Then the client must also specify the proper client key and certificate files or the server will reject the connection:

```
shell> mysql --ssl-ca=cacert.pem \
--ssl-cert=client-cert.pem \
--ssl-key=client-key.pem
```

In other words, the options are similar to those used for the server. Note that the Certificate Authority certificate has to be the same.

A client can determine whether the current connection with the server uses SSL by checking the value of the `Ssl_cipher` status variable. The value of `Ssl_cipher` is non-empty if SSL is used, and empty otherwise. For example:

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';
+-----+-----+
| Variable_name | Value               |
+-----+-----+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+-----+-----+
```

For the `mysql` client, you can use the `STATUS` or `\s` command and check the `SSL` line:

```
mysql> \s
...
SSL:                Not in use
...
```

Or:

```
mysql> \s
...
SSL:                Cipher in use is DHE-RSA-AES256-SHA
...
```

To establish a secure connection from within an application program, use the `mysql_ssl_set()` C API function to set the appropriate certificate options before calling `mysql_real_connect()`. See [Section 25.2.3.67, “mysql_ssl_set\(\)”](#). After the connection is established, you can use `mysql_get_ssl_cipher()` to determine whether SSL is in use. A non-`NULL` return value indicates a secure connection and names the SSL cipher used for encryption. A `NULL` return value indicates that SSL is not being used. See [Section 25.2.3.33, “mysql_get_ssl_cipher\(\)”](#).

5.8.7.3. SSL Command Options

The following list describes options that are used for specifying the use of SSL, certificate files, and key files. They can be given on the command line or in an option file. These options are not available unless MySQL has been built with SSL support. See [Section 5.8.7.2, “Using SSL Connections”](#). (There are also `--master-ssl*` options that can be used for setting up a secure connection from a slave replication server to a master server; see [Section 15.1.3, “Replication Options and Variables”](#).)

- `--ssl`

For the server, this option specifies that the server allows SSL connections. For a client program, it allows the client to connect to the server using SSL. This option is not sufficient in itself to cause an SSL connection to be used. You must also specify the `--ssl-ca` option, and possibly the `--ssl-cert` and `--ssl-key` options.

This option is more often used in its opposite form to override any other SSL options and indicate that SSL should *not* be used. To do this, specify the option as `--skip-ssl` or `--ssl=0`.

Note that use of `--ssl` does not *require* an SSL connection. For example, if the server or client is compiled without SSL support, a normal unencrypted connection is used.

The secure way to require use of an SSL connection is to create an account on the server that includes a `REQUIRE SSL` clause in the `GRANT` statement. Then use that account to connect to the server, where both the server and the client have SSL support enabled.

The `REQUIRE` clause allows other SSL-related restrictions as well. The description of `REQUIRE` in [Section 12.5.1.3, “GRANT Syntax”](#), provides additional detail about which SSL command options may or must be specified by clients that connect using accounts that are created using the various `REQUIRE` options.

- `--ssl-ca=file_name`

The path to a file that contains a list of trusted SSL CAs.

- `--ssl-capath=directory_name`

The path to a directory that contains trusted SSL CA certificates in PEM format.

- `--ssl-cert=file_name`

The name of the SSL certificate file to use for establishing a secure connection.

- `--ssl-cipher=cipher_list`

A list of allowable ciphers to use for SSL encryption. `cipher_list` has the same format as the `openssl ciphers` command.

Example: `--ssl-cipher=ALL:-AES:-EXP`

- `--ssl-key=file_name`

The name of the SSL key file to use for establishing a secure connection.

- `--ssl-verify-server-cert`

This option is available for client programs. It causes the server's Common Name value in the certificate that the server sends to the client to be verified against the hostname that the client uses for connecting to the server, and the connection is rejected if there is a mismatch. This feature can be used to prevent man-in-the-middle attacks. Verification is disabled by default. This option was added in MySQL 5.1.11.

As of MySQL 5.1.18, if you use SSL when establishing a client connection, you can tell the client not to authenticate the server certificate by specifying neither `--ssl-ca` nor `--ssl-capath`. The server still verifies the client according to any applicable requirements established via `GRANT` statements for the client, and it still uses any `--ssl-ca/--ssl-capath` values that were passed to server at startup time.

5.8.7.4. Setting Up SSL Certificates for MySQL

This section demonstrates how to set up SSL certificate and key files for use by MySQL servers and clients. The first example shows a simplified procedure such as you might use from the command line. The second shows a script that contains more detail. Both examples use the `openssl` command that is part of OpenSSL.

The following example shows a set of commands to create MySQL server and client certificate and key files. You will need to respond to several prompts by the `openssl` commands. For testing, you can press Enter to all prompts. For production use, you should provide non-empty responses.

```
# Create clean environment
shell> rm -rf newcerts
shell> mkdir newcerts && cd newcerts
```

```
# Create CA certificate
shell> openssl genrsa 2048 > ca-key.pem
shell> openssl req -new -x509 -nodes -days 1000 \
    -key ca-key.pem > ca-cert.pem

# Create server certificate
shell> openssl req -newkey rsa:2048 -days 1000 \
    -nodes -keyout server-key.pem > server-req.pem
shell> openssl x509 -req -in server-req.pem -days 1000 \
    -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 > server-cert.pem

# Create client certificate
shell> openssl req -newkey rsa:2048 -days 1000 \
    -nodes -keyout client-key.pem > client-req.pem
shell> openssl x509 -req -in client-req.pem -days 1000 \
    -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 > client-cert.pem
```

Here is an example script that shows how to set up SSL certificates for MySQL:

```
DIR=`pwd`/openssl
PRIV=$DIR/private

mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/cacert.pem \
    -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....++++++
# .....++++++
# writing new private key to '/home/monty/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#

openssl req -new -keyout $DIR/server-key.pem -out \
    $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..++++++
# .....++++++
# writing new private key to '/home/monty/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
```

```

# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key (optional)
#

openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca -policy policy_anything -out $DIR/server-cert.pem \
-config $DIR/openssl.cnf -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName             :PRINTABLE:'FI'
# organizationName        :PRINTABLE:'MySQL AB'
# commonName               :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
$DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....++++++
# .....++++++
# writing new private key to '/home/monty/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove a passphrase from the key (optional)
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#
openssl ca -policy policy_anything -out $DIR/client-cert.pem \
-config $DIR/openssl.cnf -infiles $DIR/client-req.pem

# Sample output:

```

```
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName             :PRINTABLE:'FI'
# organizationName        :PRINTABLE:'MySQL AB'
# commonName              :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cnf=""
cnf="$cnf [client]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/client-cert.pem"
cnf="$cnf ssl-key=$DIR/client-key.pem"
cnf="$cnf [mysqld]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/server-cert.pem"
cnf="$cnf ssl-key=$DIR/server-key.pem"
echo $cnf | replace " " ' '
' > $DIR/my.cnf
```

To test SSL connections, start the server as follows, where `$DIR` is the pathname to the directory where the sample `my.cnf` option file is located:

```
shell> mysqld --defaults-file=$DIR/my.cnf &
```

Then invoke a client program using the same option file:

```
shell> mysql --defaults-file=$DIR/my.cnf
```

If you have a MySQL source distribution, you can also test your setup by modifying the preceding `my.cnf` file to refer to the demonstration certificate and key files in the `mysql-test/std_data` directory of the distribution.

5.8.7.5. Connecting to MySQL Remotely from Windows with SSH

Here is a note that describes how to get a secure connection to a remote MySQL server with SSH (by David Carlson dcarlson@mplcomm.com):

1. Install an SSH client on your Windows machine. As a user, the best non-free one I have found is from [SecureCRT](http://www.vandyke.com/) from <http://www.vandyke.com/>. Another option is [f-secure](http://www.f-secure.com/) from <http://www.f-secure.com/>. You can also find some free ones on Google at http://directory.google.com/Top/Computers/Security/Products_and_Tools/Cryptography/SSH/Clients/Windows/.
2. Start your Windows SSH client. Set `Host_Name` = `yourmysqlserver_URL_or_IP`. Set `userid=your_userid` to log in to your server. This `userid` value might not be the same as the username of your MySQL account.
3. Set up port forwarding. Either do a remote forward (Set `local_port`: `3306`, `remote_host`: `yourmysqlserver-name_or_ip`, `remote_port`: `3306`) or a local forward (Set `port`: `3306`, `host`: `localhost`, `remote port`: `3306`).
4. Save everything, otherwise you will have to redo it the next time.
5. Log in to your server with the SSH session you just created.
6. On your Windows machine, start some ODBC application (such as Access).
7. Create a new file in Windows and link to MySQL using the ODBC driver the same way you normally do, except type in `localhost` for the MySQL host server, not `yourmysqlservername`.

At this point, you should have an ODBC connection to MySQL, encrypted using SSH.

5.9. Backup and Recovery

This section discusses how to make database backups (full and incremental) and how to perform table maintenance. The syntax of the SQL statements described here is given in [Chapter 12, SQL Statement Syntax](#). Much of the information here pertains primarily to **MyISAM** tables. Additional information about **InnoDB** backup procedures is given in [Section 13.5.8, “Backing Up and Recovering an InnoDB Database”](#).

5.9.1. Database Backups

Because MySQL tables are stored as files, it is easy to do a backup. To get a consistent backup, do a **LOCK TABLES** on the relevant tables, followed by **FLUSH TABLES** for the tables. See [Section 12.4.5, “LOCK TABLES and UNLOCK TABLES Syntax”](#), and [Section 12.5.5.2, “FLUSH Syntax”](#). You need only a read lock; this allows other clients to continue to query the tables while you are making a copy of the files in the database directory. The **FLUSH TABLES** statement is needed to ensure that the all active index pages are written to disk before you start the backup.

To make an SQL-level backup of a table, you can use **SELECT INTO ... OUTFILE**. For this statement, the output file cannot already exist because allowing files to be overwritten would constitute a security risk. See [Section 12.2.7, “SELECT Syntax”](#).

Another technique for backing up a database is to use the **mysqldump** program or the **mysqlhotcopy** script. See [Section 7.13, “mysqldump — A Database Backup Program”](#), and [Section 7.14, “mysqlhotcopy — A Database Backup Program”](#).

1. Create a full backup of your database:

```
shell> mysqldump --tab=/path/to/some/dir --opt db_name
```

Or:

```
shell> mysqlhotcopy db_name /path/to/some/dir
```

You can also create a binary backup simply by copying all table files (***.frm**, ***.MYD**, and ***.MYI** files), as long as the server isn't updating anything. The **mysqlhotcopy** script uses this method. (But note that these methods do not work if your database contains **InnoDB** tables. **InnoDB** does not store table contents in database directories, and **mysqlhotcopy** works only for **MyISAM** tables.)

2. Stop **mysqld** if it is running, then start it with the **--log-bin[=file_name]** option. See [Section 5.11.4, “The Binary Log”](#). The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you executed **mysqldump**.

For **InnoDB** tables, it is possible to perform an online backup that takes no locks on tables; see [Section 7.13, “mysqldump — A Database Backup Program”](#).

MySQL supports incremental backups: You need to start the server with the **--log-bin** option to enable binary logging; see [Section 5.11.4, “The Binary Log”](#). At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using **FLUSH LOGS**. This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained further below. The next time you do a full backup, you should also rotate the binary log using **FLUSH LOGS**, **mysqldump --flush-logs**, or **mysqlhotcopy --flushlog**. See [Section 7.13, “mysqldump — A Database Backup Program”](#), and [Section 7.14, “mysqlhotcopy — A Database Backup Program”](#).

If your MySQL server is a slave replication server, then regardless of the backup method you choose, you should also back up the **master.info** and **relay-log.info** files when you back up your slave's data. These files are always needed to resume replication after you restore the slave's data. If your slave is subject to replicating **LOAD DATA INFILE** commands, you should also back up any **SQL_LOAD-*** files that may exist in the directory specified by the **--slave-load-tmpdir** option. (This location defaults to the value of the **tmpdir** variable if not specified.) The slave needs these files to resume replication of any interrupted **LOAD DATA INFILE** operations.

MySQL Enterprise

The MySQL Enterprise Monitor provides numerous advisors that issue immediate warnings should replication issues arise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

If you have to restore **MyISAM** tables, try to recover them using **REPAIR TABLE** or **myisamchk -r** first. That should work in 99.9% of all cases. If **myisamchk** fails, try the following procedure. Note that it works only if you have enabled binary logging by

starting MySQL with the `--log-bin` option.

1. Restore the original `mysqldump` backup, or binary backup.
2. Execute the following command to re-run the updates in the binary logs:

```
shell> mysqlbinlog binlog.[0-9]* | mysql
```

In some cases, you may want to re-run only certain binary logs, from certain positions (usually you want to re-run all binary logs from the date of the restored backup, excepting possibly some incorrect statements). See [Section 7.11, “mysqlbinlog — Utility for Processing Binary Log Files”](#), for more information on the `mysqlbinlog` utility and how to use it.

You can also make selective backups of individual files:

- To dump the table, use `SELECT * INTO OUTFILE 'file_name' FROM tbl_name`.
- To reload the table, use `LOAD DATA INFILE 'file_name' REPLACE ...`. To avoid duplicate rows, the table must have a `PRIMARY KEY` or a `UNIQUE` index. The `REPLACE` keyword causes old rows to be replaced with new ones when a new row duplicates an old row on a unique key value.

If you have performance problems with your server while making backups, one strategy that can help is to set up replication and perform backups on the slave rather than on the master. See [Chapter 15, Replication](#).

If you are using a Veritas filesystem, you can make a backup like this:

1. From a client program, execute `FLUSH TABLES WITH READ LOCK`.
2. From another shell, execute `mount vxfs snapshot`.
3. From the first client, execute `UNLOCK TABLES`.
4. Copy files from the snapshot.
5. Unmount the snapshot.

5.9.2. Example Backup and Recovery Strategy

This section discusses a procedure for performing backups that allows you to recover data after several types of crashes:

- Operating system crash
- Power failure
- Filesystem crash
- Hardware problem (hard drive, motherboard, and so forth)

The example commands do not include options such as `--user` and `--password` for the `mysqldump` and `mysql` programs. You should include such options as necessary so that the MySQL server allows you to connect to it.

We assume that data is stored in the `InnoDB` storage engine, which has support for transactions and automatic crash recovery. We also assume that the MySQL server is under load at the time of the crash. If it were not, no recovery would ever be needed.

For cases of operating system crashes or power failures, we can assume that MySQL's disk data is available after a restart. The `InnoDB` data files might not contain consistent data due to the crash, but `InnoDB` reads its logs and finds in them the list of pending committed and non-committed transactions that have not been flushed to the data files. `InnoDB` automatically rolls back those transactions that were not committed, and flushes to its data files those that were committed. Information about this recovery process is conveyed to the user through the MySQL error log. The following is an example log excerpt:

```

InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections

```

For the cases of filesystem crashes or hardware problems, we can assume that the MySQL disk data is *not* available after a restart. This means that MySQL fails to start successfully because some blocks of disk data are no longer readable. In this case, it is necessary to re-format the disk, install a new one, or otherwise correct the underlying problem. Then it is necessary to recover our MySQL data from backups, which means that we must already have made backups. To make sure that is the case, we should design a backup policy.

5.9.2.1. Backup Policy

We all know that backups must be scheduled periodically. A full backups (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, **InnoDB Hot Backup** provides online non-blocking physical backup of the **InnoDB** data files, and **mysqldump** provides online logical backup. This discussion uses **mysqldump**.

MySQL Enterprise

For expert advice on backups and replication, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

Assume that we make a backup on Sunday at 1 p.m., when load is low. The following command makes a full backup of all our **InnoDB** tables in all databases:

```
shell> mysqldump --single-transaction --all-databases > backup_sunday_1_PM.sql
```

This is an online, non-blocking backup that does not disturb the reads and writes on the tables. We assumed earlier that our tables are **InnoDB** tables, so **--single-transaction** uses a consistent read and guarantees that data seen by **mysqldump** does not change. (Changes made by other clients to **InnoDB** tables are not seen by the **mysqldump** process.) If we do also have other types of tables, we must assume that they are not changed during the backup. For example, for the **MyISAM** tables in the **mysql** database, we must assume that no administrative changes are being made to MySQL accounts during the backup.

The resulting **.sql** file produced by **mysqldump** contains a set of SQL **INSERT** statements that can be used to reload the dumped tables at a later time.

Full backups are necessary, but they are not always convenient. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that has not changed since the previous full backup. After we have made the initial full backup, it is more efficient to make incremental backups. They are smaller and take less time to produce. The tradeoff is that, at recovery time, you cannot restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.

To make incremental backups, we need to save the incremental changes. The MySQL server should always be started with the **-log-bin** option so that it stores these changes in a file while it updates data. This option enables binary logging, so that the server writes each SQL statement that updates data into a file called a MySQL binary log. Looking at the data directory of a MySQL server that was started with the **--log-bin** option and that has been running for some days, we find these MySQL binary log files:

```

-rw-rw---- 1 guilhem guilhem 1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem guilhem      4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem guilhem    79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem guilhem   508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem guilhem 2247446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem guilhem 998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem guilhem   361 Nov 14 10:07 gbichot2-bin.index

```

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is run-

ning, you can also tell it to close the current binary log file and begin a new one manually by issuing a `FLUSH LOGS` SQL statement or with a `mysqladmin flush-logs` command. `mysqldump` also has an option to flush the logs. The `.index` file in the data directory contains the list of all MySQL binary logs in the directory. This file is used for replication.

The MySQL binary logs are important for recovery because they form the set of incremental backups. If you make sure to flush the logs when you make your full backup, then any binary log files created afterward contain all the data changes made since the backup. Let's modify the previous `mysqldump` command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so that the dump file contains the name of the new current binary log:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
--all-databases > backup_sunday_1_PM.sql
```

After executing this command, the data directory contains a new binary log file, `gbichot2-bin.000007`. The resulting `.sql` file includes these lines:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',»
MASTER_LOG_POS=4;
```

Because the `mysqldump` command made a full backup, those lines mean two things:

- The `.sql` file contains all changes made before any changes written to the `gbichot2-bin.000007` binary log file or newer.
- All data changes logged after the backup are not present in the `.sql`, but are present in the `gbichot2-bin.000007` binary log file or newer.

On Monday at 1 p.m., we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a `mysqladmin flush-logs` command creates `gbichot2-bin.000008`. All changes between the Sunday 1 p.m. full backup and Monday 1 p.m. will be in the `gbichot2-bin.000007` file. This incremental backup is important, so it is a good idea to copy it to a safe place. (For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday at 1 p.m., execute another `mysqladmin flush-logs` command. All changes between Monday 1 p.m. and Tuesday 1 p.m. will be in the `gbichot2-bin.000008` file (which also should be copied somewhere safe).

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
--all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

Note

Deleting the MySQL binary logs with `mysqldump --delete-master-logs` can be dangerous if your server is a replication master server, because slave servers might not yet fully have processed the contents of the binary log. The description for the `PURGE MASTER LOGS` statement explains what should be verified before deleting the MySQL binary logs. See [Section 12.6.1.1, “PURGE MASTER LOGS Syntax”](#).

5.9.2.2. Using Backups for Recovery

Now, suppose that we have a catastrophic crash on Wednesday at 8 a.m. that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 p.m.). The full backup file is just a set of SQL statements, so restoring it is very easy:

```
shell> mysql < backup_sunday_1_PM.sql
```

At this point, the data is restored to its state as of Sunday 1 p.m.. To restore the changes made since then, we must use the incremental backups; that is, the `gbichot2-bin.000007` and `gbichot2-bin.000008` binary log files. Fetch the files if necessary from where they were backed up, and then process their contents like this:

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

We now have recovered the data to its state as of Tuesday 1 p.m., but still are missing the changes from that date to the date of the crash. To not lose them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID

disks, SAN, ...) different from the place where it stores its data files, so that these logs were not on the destroyed disk. (That is, we can start the server with a `--log-bin` option that specifies a location on a different physical device from the one on which the data directory resides. That way, the logs are safe even if the device containing the directory is lost.) If we had done this, we would have the `gbi-chot2-bin.000009` file at hand, and we could apply it using `mysqlbinlog` and `mysql` to restore the most recent data changes with no loss up to the moment of the crash.

5.9.2.3. Backup Strategy Summary

In case of an operating system crash or power failure, `InnoDB` itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

- Always run the MySQL server with the `--log-bin` option, or even `--log-bin=log_name`, where the log file name is located on some safe media different from the drive on which the data directory is located. If you have such safe media, this technique can also be good for disk load balancing (which results in a performance improvement).
- Make periodic full backups, using the `mysqldump` command shown earlier in Section 5.9.2.1, “Backup Policy”, that makes an on-line, non-blocking backup.
- Make periodic incremental backups by flushing the logs with `FLUSH LOGS` or `mysqladmin flush-logs`.

5.9.3. Point-in-Time Recovery

If a MySQL server was started with the `--log-bin` option to enable binary logging, you can use the `mysqlbinlog` utility to recover data from the binary log files, starting from a specified point in time (for example, since your last backup) until the present or another specified point in time. For information on enabling the binary log and using `mysqlbinlog`, see Section 5.11.4, “The Binary Log”, and Section 7.11, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

MySQL Enterprise

For maximum data recovery, MySQL Enterprise Monitor advises subscribers to synchronize to disk at each write. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

To restore data from a binary log, you must know the location and name of the current binary log file. By default, the server creates binary log files in the data directory, but a pathname can be specified with the `--log-bin` option to place the files in a different location. Typically the option is given in an option file (that is, `my.cnf` or `my.ini`, depending on your system). It can also be given on the command line when the server is started. To determine the name of the current binary log file, issue the following statement:

```
mysql> SHOW MASTER STATUS
```

If you prefer, you can execute the following command from the command line instead:

```
shell> mysql -u root -p -E -e "SHOW MASTER STATUS"
```

Enter the `root` password for your server when `mysql` prompts you for it.

To view the contents of a binary log, use `mysqlbinlog`. See Section 7.11, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

5.9.3.1. Specifying Times for Recovery

To indicate the start and end times for recovery, specify the `--start-date` and `--stop-date` options for `mysqlbinlog`, in `DATETIME` format. As an example, suppose that exactly at 10:00 a.m. on April 20, 2005 an SQL statement was executed that deleted a large table. To restore the table and data, you could restore the previous night's backup, and then execute the following command:

```
shell> mysqlbinlog --stop-date="2005-04-20 9:59:59" \
/var/log/mysql/bin.123456 | mysql -u root -p
```

This command recovers all of the data up until the date and time given by the `--stop-date` option. If you did not detect the erroneous SQL statement that was entered until hours later, you will probably also want to recover the activity that occurred afterward. Based on this, you could run `mysqlbinlog` again with a start date and time, like so:

```
shell> mysqlbinlog --start-date="2005-04-20 10:01:00" \
/var/log/mysql/bin.123456 | mysql -u root -p
```

In this command, the SQL statements logged from 10:01 a.m. on will be re-executed. The combination of restoring of the previous night's dump file and the two `mysqlbinlog` commands restores everything up until one second before 10:00 a.m. and everything from 10:01 a.m. on. You should examine the log to be sure of the exact times to specify for the commands. To display the log file contents without executing them, use this command:

```
shell> mysqlbinlog /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

Then open the file with a text editor to examine it.

5.9.3.2. Specifying Positions for Recovery

Instead of specifying dates and times, the `--start-position` and `--stop-position` options for `mysqlbinlog` can be used for specifying log positions. They work the same as the start and stop date options, except that you specify log position numbers rather than dates. Using positions may enable you to be more precise about which part of the log to recover, especially if many transactions occurred around the same time as a damaging SQL statement. To determine the position numbers, run `mysqlbinlog` for a range of times near the time when the unwanted transaction was executed, but redirect the results to a text file for examination. This can be done like so:

```
shell> mysqlbinlog --start-date="2005-04-20 9:55:00" \
--stop-date="2005-04-20 10:05:00" \
/var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

This command creates a small text file in the `/tmp` directory that contains the SQL statements around the time that the deleterious SQL statement was executed. Open this file with a text editor and look for the statement that you don't want to repeat. Determine the positions in the binary log for stopping and resuming the recovery and make note of them. Positions are labeled as `log_pos` followed by a number. After restoring the previous backup file, use the position numbers to process the binary log file. For example, you would use commands something like these:

```
shell> mysqlbinlog --stop-position="368312" /var/log/mysql/bin.123456 \
| mysql -u root -p
shell> mysqlbinlog --start-position="368315" /var/log/mysql/bin.123456 \
| mysql -u root -p
```

The first command recovers all the transactions up until the stop position given. The second command recovers all transactions from the starting position given until the end of the binary log. Because the output of `mysqlbinlog` includes `SET TIMESTAMP` statements before each SQL statement recorded, the recovered data and related MySQL logs will reflect the original times at which the transactions were executed.

5.9.4. Table Maintenance and Crash Recovery

This section discusses how to use `myisamchk` to check or repair `MyISAM` tables (tables that have `.MYD` and `.MYI` files for storing data and indexes). For general `myisamchk` background, see [Section 7.5, “myisamchk — MyISAM Table-Maintenance Utility”](#).

You can use `myisamchk` to get information about your database tables or to check, repair, or optimize them. The following sections describe how to perform these operations and how to set up a table maintenance schedule.

Even though table repair with `myisamchk` is quite secure, it is always a good idea to make a backup *before* doing a repair or any maintenance operation that could make a lot of changes to a table.

`myisamchk` operations that affect indexes can cause `FULLTEXT` indexes to be rebuilt with full-text parameters that are incompatible with the values used by the MySQL server. To avoid this problem, follow the guidelines in [Section 7.5.1, “myisamchk General Options”](#).

In many cases, you may find it simpler to do `MyISAM` table maintenance using the SQL statements that perform operations that `myisamchk` can do:

- To check or repair `MyISAM` tables, use `CHECK TABLE` or `REPAIR TABLE`.
- To optimize `MyISAM` tables, use `OPTIMIZE TABLE`.
- To analyze `MyISAM` tables, use `ANALYZE TABLE`.

These statements can be used directly or by means of the `mysqlcheck` client program. One advantage of these statements over `myisamchk` is that the server does all the work. With `myisamchk`, you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between `myisamchk` and the server. See [Section 12.5.2.1, “ANALYZE TABLE Syntax”](#), [Section 12.5.2.3, “CHECK TABLE Syntax”](#), [Section 12.5.2.5, “OPTIMIZE TABLE Syntax”](#), and [Section 12.5.2.6, “REPAIR TABLE Syntax”](#).

5.9.4.1. Using `myisamchk` for Crash Recovery

This section describes how to check for and deal with data corruption in MySQL databases. If your tables become corrupted frequently, you should try to find the reason why. See [Section B.1.4.2, “What to Do If MySQL Keeps Crashing”](#).

For an explanation of how `MyISAM` tables can become corrupted, see [Section 13.4.4, “MyISAM Table Problems”](#).

If you run `mysqld` with external locking disabled (which is the default as of MySQL 4.0), you cannot reliably use `myisamchk` to check a table when `mysqld` is using the same table. If you can be certain that no one will access the tables through `mysqld` while you run `myisamchk`, you only have to execute `mysqladmin flush-tables` before you start checking the tables. If you cannot guarantee this, you must stop `mysqld` while you check the tables. If you run `myisamchk` to check tables that `mysqld` is updating at the same time, you may get a warning that a table is corrupt even when it is not.

If the server is run with external locking enabled, you can use `myisamchk` to check tables at any time. In this case, if the server tries to update a table that `myisamchk` is using, the server will wait for `myisamchk` to finish before it continues.

If you use `myisamchk` to repair or optimize tables, you *must* always ensure that the `mysqld` server is not using the table (this also applies if external locking is disabled). If you don't stop `mysqld`, you should at least do a `mysqladmin flush-tables` before you run `myisamchk`. Your tables *may become corrupted* if the server and `myisamchk` access the tables simultaneously.

When performing crash recovery, it is important to understand that each `MyISAM` table `tbl_name` in a database corresponds to three files in the database directory:

File	Purpose
<code>tbl_name.frm</code>	Definition (format) file
<code>tbl_name.MYD</code>	Data file
<code>tbl_name.MYI</code>	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

`myisamchk` works by creating a copy of the `.MYD` data file row by row. It ends the repair stage by removing the old `.MYD` file and renaming the new file to the original file name. If you use `--quick`, `myisamchk` does not create a temporary `.MYD` file, but instead assumes that the `.MYD` file is correct and generates only a new index file without touching the `.MYD` file. This is safe, because `myisamchk` automatically detects whether the `.MYD` file is corrupt and aborts the repair if it is. You can also specify the `--quick` option twice to `myisamchk`. In this case, `myisamchk` does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the `.MYD` file. Normally the use of two `--quick` options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running `myisamchk`.

5.9.4.2. How to Check `MyISAM` Tables for Errors

To check a `MyISAM` table, use the following commands:

- `myisamchk tbl_name`

This finds 99.99% of all errors. What it cannot find is corruption that involves *only* the data file (which is very unusual). If you want to check a table, you should normally run `myisamchk` without options or with the `-s` (silent) option.

- `myisamchk -m tbl_name`

This finds 99.999% of all errors. It first checks all index entries for errors and then reads through all rows. It calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

- `myisamchk -e tbl_name`

This does a complete and thorough check of all data (`-e` means “extended check”). It does a check-read of every key for each row

to verify that they indeed point to the correct row. This may take a long time for a large table that has many indexes. Normally, `myisamchk` stops after the first error it finds. If you want to obtain more information, you can add the `-v` (verbose) option. This causes `myisamchk` to keep going, up through a maximum of 20 errors.

- `myisamchk -e -i tbl_name`

This is like the previous command, but the `-i` option tells `myisamchk` to print additional statistical information.

In most cases, a simple `myisamchk` command with no arguments other than the table name is sufficient to check a table.

5.9.4.3. How to Repair Tables

The discussion in this section describes how to use `myisamchk` on `MyISAM` tables (extensions `.MYI` and `.MYD`).

You can also (and should, if possible) use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See [Section 12.5.2.3, “CHECK TABLE Syntax”](#), and [Section 12.5.2.6, “REPAIR TABLE Syntax”](#).

Symptoms of corrupted tables include queries that abort unexpectedly and observable errors such as these:

- `tbl_name.frm` is locked against change
- Can't find file `tbl_name.MYI` (Errcode: `nnn`)
- Unexpected end of file
- Record file is crashed
- Got error `nnn` from table handler

To get more information about the error, run `pererror nnn`, where `nnn` is the error number. The following example shows how to use `pererror` to find the meanings for the most common error numbers that indicate a problem with a table:

```
shell> pererror 126 127 132 134 135 136 141 144 145
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 132 = Old database file
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 135 = No more room in record file
MySQL error code 136 = No more room in index file
MySQL error code 141 = Duplicate unique key or constraint on write or update
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Note that error 135 (no more room in record file) and error 136 (no more room in index file) are not errors that can be fixed by a simple repair. In this case, you must use `ALTER TABLE` to increase the `MAX_ROWS` and `AVG_ROW_LENGTH` table option values:

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

If you do not know the current table option values, use `SHOW CREATE TABLE`.

For the other errors, you must repair your tables. `myisamchk` can usually detect and fix most problems that occur.

The repair process involves up to four stages, described here. Before you begin, you should change location to the database directory and check the permissions of the table files. On Unix, make sure that they are readable by the user that `mysqld` runs as (and to you, because you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

This section is for the cases where a table check fails (such as those described in [Section 5.9.4.2, “How to Check MyISAM Tables for Errors”](#)), or you want to use the extended features that `myisamchk` provides.

The options that you can use for table maintenance with `myisamchk` are described in [Section 7.5, “myisamchk — MyISAM Table-Maintenance Utility”](#).

If you are going to repair a table from the command line, you must first stop the `mysqld` server. Note that when you do `mysqladmin shutdown` on a remote server, the `mysqld` server is still alive for a while after `mysqladmin` returns, until all statement-processing

has stopped and all index changes have been flushed to disk.

Stage 1: Checking your tables

Run `myisamchk *.MYI` or `myisamchk -e *.MYI` if you have more time. Use the `-s` (silent) option to suppress unnecessary information.

If the `mysqld` server is stopped, you should use the `--update-state` option to tell `myisamchk` to mark the table as “checked.”

You have to repair only those tables for which `myisamchk` announces an error. For such tables, proceed to Stage 2.

If you get unexpected errors when checking (such as `out of memory` errors), or if `myisamchk` crashes, go to Stage 3.

Stage 2: Easy safe repair

First, try `myisamchk -r -q tbl_name` (`-r -q` means “quick recovery mode”). This attempts to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work, and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use `myisamchk -r tbl_name` (`-r` means “recovery mode”). This removes incorrect rows and deleted rows from the data file and reconstructs the index file.
3. If the preceding step fails, use `myisamchk --safe-recover tbl_name`. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower).

Note: If you want a repair operation to go much faster, you should set the values of the `sort_buffer_size` and `key_buffer_size` variables each to about 25% of your available memory when running `myisamchk`.

If you get unexpected errors when repairing (such as `out of memory` errors), or if `myisamchk` crashes, go to Stage 3.

Stage 3: Difficult repair

You should reach this stage only if the first 16KB block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it is necessary to create a new index file. Do so as follows:

1. Move the data file to a safe place.
2. Use the table description file to create new (empty) data and index files:

```
shell> mysql db_name
mysql> SET AUTOCOMMIT=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

3. Copy the old data file back onto the newly created data file. (Do not just move the old file back onto the new file. You want to retain a copy in case something goes wrong.)

Go back to Stage 2. `myisamchk -r -q` should work. (This should not be an endless loop.)

You can also use the `REPAIR TABLE tbl_name USE_FRM` SQL statement, which performs the whole procedure automatically. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `REPAIR TABLE`. See [Section 12.5.2.6, “REPAIR TABLE Syntax”](#).

Stage 4: Very difficult repair

You should reach this stage only if the `.frm` description file has also crashed. That should never happen, because the description file is not changed after the table is created:

1. Restore the description file from a backup and go back to Stage 3. You can also restore the index file and go back to Stage 2. In the latter case, you should start with `myisamchk -r`.

2. If you do not have a backup but know exactly how the table was created, create a copy of the table in another database. Remove the new data file, and then move the `.frm` description and `.MYI` index files from the other database to your crashed database. This gives you new description and index files, but leaves the `.MYD` data file alone. Go back to Stage 2 and attempt to reconstruct the index file.

5.9.4.4. Table Optimization

To coalesce fragmented rows and eliminate wasted space that results from deleting or updating rows, run `myisamchk` in recovery mode:

```
shell> myisamchk -r tbl_name
```

You can optimize a table in the same way by using the `OPTIMIZE TABLE` SQL statement. `OPTIMIZE TABLE` does a table repair and a key analysis, and also sorts the index tree so that key lookups are faster. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `OPTIMIZE TABLE`. See [Section 12.5.2.5, “OPTIMIZE TABLE Syntax”](#).

`myisamchk` has a number of other options that you can use to improve the performance of a table:

- `--analyze, -a`
- `--sort-index, -S`
- `--sort-records=index_num, -R index_num`

For a full description of all available options, see [Section 7.5, “myisamchk — MyISAM Table-Maintenance Utility”](#).

5.9.4.5. Getting Information About a Table

To obtain a description of a table or statistics about it, use the commands shown here. We explain some of the information in more detail later.

- `myisamchk -d tbl_name`

Runs `myisamchk` in “describe mode” to produce a description of your table. If you start the MySQL server with external locking disabled, `myisamchk` may report an error for a table that is updated while it runs. However, because `myisamchk` does not change the table in describe mode, there is no risk of destroying data.

- `myisamchk -d -v tbl_name`

Adding `-v` runs `myisamchk` in verbose mode so that it produces more information about what it is doing.

- `myisamchk -eis tbl_name`

Shows only the most important information from a table. This operation is slow because it must read the entire table.

- `myisamchk -eiv tbl_name`

This is like `-eis`, but tells you what is being done.

The `tbl_name` argument can be either the name of a `MyISAM` table or the name of its index file, as described in [Section 7.5, “myisamchk — MyISAM Table-Maintenance Utility”](#). Multiple `tbl_name` arguments can be given.

Sample output for some of these commands follows. They are based on a table with these data and index file sizes:

```
-rw-rw-r-- 1 monty tcx 317235748 Jan 12 17:30 company.MYD
-rw-rw-r-- 1 davida tcx 96482304 Jan 12 18:35 company.MYI
```

Example of `myisamchk -d` output:

```
MyISAM file: company.MYI
```

```
Record format: Fixed length
Data records: 1403698 Deleted blocks: 0
Recordlength: 226
```

```
table description:
Key Start Len Index Type
1 2 8 unique double
2 15 10 multip. text packed stripped
3 219 8 multip. double
4 63 10 multip. text packed stripped
5 167 2 multip. unsigned short
6 177 4 multip. unsigned long
7 155 4 multip. text
8 138 4 multip. unsigned long
9 177 4 multip. unsigned long
193 1 text
```

Example of `myisamchk -d -v` output:

```
MyISAM file: company
Record format: Fixed length
File-version: 1
Creation time: 1999-10-30 12:12:51
Recover time: 1999-10-31 19:13:01
Status: checked
Data records: 1403698 Deleted blocks: 0
Datafile parts: 1403698 Deleted data: 0
Datafile pointer (bytes): 3 Keyfile pointer (bytes): 3
Max datafile length: 3791650815 Max keyfile length: 4294967294
Recordlength: 226
```

```
table description:
Key Start Len Index Type Rec/key Root Blocksize
1 2 8 unique double 1 15845376 1024
2 15 10 multip. text packed stripped 2 25062400 1024
3 219 8 multip. double 73 40907776 1024
4 63 10 multip. text packed stripped 5 48097280 1024
5 167 2 multip. unsigned short 4840 55200768 1024
6 177 4 multip. unsigned long 1346 65145856 1024
7 155 4 multip. text 4995 75090944 1024
8 138 4 multip. unsigned long 87 85036032 1024
9 177 4 multip. unsigned long 178 96481280 1024
193 1 text
```

Example of `myisamchk -eis` output:

```
Checking MyISAM file: company
Key: 1: Keyblocks used: 97% Packed: 0% Max levels: 4
Key: 2: Keyblocks used: 98% Packed: 50% Max levels: 4
Key: 3: Keyblocks used: 97% Packed: 0% Max levels: 4
Key: 4: Keyblocks used: 99% Packed: 60% Max levels: 3
Key: 5: Keyblocks used: 99% Packed: 0% Max levels: 3
Key: 6: Keyblocks used: 99% Packed: 0% Max levels: 3
Key: 7: Keyblocks used: 99% Packed: 0% Max levels: 3
Key: 8: Keyblocks used: 99% Packed: 0% Max levels: 3
Key: 9: Keyblocks used: 98% Packed: 0% Max levels: 4
Total: Keyblocks used: 98% Packed: 17%

Records: 1403698 M.recordlength: 226
Packed: 0%
Recordspace used: 100% Empty space: 0%
Blocks/Record: 1.00
Record blocks: 1403698 Delete blocks: 0
Recorddata: 317235748 Deleted data: 0
Lost space: 0 Linkdata: 0

User time 1626.51, System time 232.36
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 627, Swaps 0
Blocks in 0 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 639, Involuntary context switches 28966
```

Example of `myisamchk -eiv` output:

```
Checking MyISAM file: company
Data records: 1403698 Deleted blocks: 0
- check file-size
- check delete-chain
block_size 1024:
index 1:
index 2:
index 3:
index 4:
index 5:
```

```

index 6:
index 7:
index 8:
index 9:
No recordlinks
- check index reference
- check data record references index: 1
Key: 1: Keyblocks used: 97% Packed: 0% Max levels: 4
- check data record references index: 2
Key: 2: Keyblocks used: 98% Packed: 50% Max levels: 4
- check data record references index: 3
Key: 3: Keyblocks used: 97% Packed: 0% Max levels: 4
- check data record references index: 4
Key: 4: Keyblocks used: 99% Packed: 60% Max levels: 3
- check data record references index: 5
Key: 5: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 6
Key: 6: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 7
Key: 7: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 8
Key: 8: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 9
Key: 9: Keyblocks used: 98% Packed: 0% Max levels: 4
Total: Keyblocks used: 9% Packed: 17%

- check records and index references
*** LOTS OF ROW NUMBERS DELETED ***

Records: 1403698 M.recordlength: 226 Packed: 0%
Recordspace used: 100% Empty space: 0% Blocks/Record: 1.00
Record blocks: 1403698 Delete blocks: 0
Recorddata: 317235748 Deleted data: 0
Lost space: 0 Linkdata: 0

User time 1639.63, System time 251.61
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 10580, Swaps 0
Blocks in 4 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 10604, »
Involuntary context switches 122798

```

Explanations for the types of information `myisamchk` produces are given here. “Keyfile” refers to the index file. “Record” and “row” are synonymous.

- **MyISAM file**

Name of the **MyISAM** (index) file.

- **File-version**

Version of **MyISAM** format. Currently always 2.

- **Creation time**

When the data file was created.

- **Recover time**

When the index/data file was last reconstructed.

- **Data records**

How many rows are in the table.

- **Deleted blocks**

How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See [Section 5.9.4.4, “Table Optimization”](#).

- **Datafile parts**

For dynamic-row format, this indicates how many data blocks there are. For an optimized table without fragmented rows, this is the same as **Data records**.

- Deleted data

How many bytes of unreclaimed deleted data there are. You can optimize your table to minimize this space. See [Section 5.9.4.4, “Table Optimization”](#).

- Datafile pointer

The size of the data file pointer, in bytes. It is usually 2, 3, 4, or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from MySQL yet. For fixed tables, this is a row address. For dynamic tables, this is a byte address.

- Keyfile pointer

The size of the index file pointer, in bytes. It is usually 1, 2, or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by MySQL. It is always a block address.

- Max datafile length

How long the table data file can become, in bytes.

- Max keyfile length

How long the table index file can become, in bytes.

- Recordlength

How much space each row takes, in bytes.

- Record format

The format used to store table rows. The preceding examples use [Fixed length](#). Other possible values are [Compressed](#) and [Packed](#).

- table description

A list of all keys in the table. For each key, [myisamchk](#) displays some low-level information:

- Key

This key's number.

- Start

Where in the row this portion of the index starts.

- Len

How long this portion of the index is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column.

- Index

Whether a key value can exist multiple times in the index. Possible values are [unique](#) or [multip.](#) (multiple).

- Type

What data type this portion of the index has. This is a [MyISAM](#) data type with the possible values [packed](#), [stripped](#), or [empty](#).

- Root

Address of the root index block.

- Blocksize

The size of each index block. By default this is 1024, but the value may be changed at compile time when MySQL is built from source.

- **Rec/key**

This is a statistical value used by the optimizer. It tells how many rows there are per value for this index. A unique index always has a value of 1. This may be updated after a table is loaded (or greatly changed) with `myisamchk -a`. If this is not updated at all, a default value of 30 is given.

For the table shown in the examples, there are two **table description** lines for the ninth index. This indicates that it is a multiple-part index with two parts.

- **Keyblocks used**

What percentage of the keyblocks are used. When a table has just been reorganized with `myisamchk`, as for the table in the examples, the values are very high (very near the theoretical maximum).

- **Packed**

MySQL tries to pack key values that have a common suffix. This can only be used for indexes on **CHAR** and **VARCHAR** columns. For long indexed strings that have similar leftmost parts, this can significantly reduce the space used. In the third of the preceding examples, the fourth key is 10 characters long and a 60% reduction in space is achieved.

- **Max levels**

How deep the B-tree for this key is. Large tables with long key values get high values.

- **Records**

How many rows are in the table.

- **M.recordlength**

The average row length. This is the exact row length for tables with fixed-length rows, because all rows have the same length.

- **Packed**

MySQL strips spaces from the end of strings. The **Packed** value indicates the percentage of savings achieved by doing this.

- **Recordspace used**

What percentage of the data file is used.

- **Empty space**

What percentage of the data file is unused.

- **Blocks/Record**

Average number of blocks per row (that is, how many links a fragmented row is composed of). This is always 1.0 for fixed-format tables. This value should stay as close to 1.0 as possible. If it gets too large, you can reorganize the table. See [Section 5.9.4.4, “Table Optimization”](#).

- **Recordblocks**

How many blocks (links) are used. For fixed-format tables, this is the same as the number of rows.

- **Deleteblocks**

How many blocks (links) are deleted.

- **Recorddata**

How many bytes in the data file are used.

- **Deleted data**

How many bytes in the data file are deleted (unused).

- **Lost space**

If a row is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

- **Linkdata**

When the dynamic table format is used, row fragments are linked with pointers (4 to 7 bytes each). **Linkdata** is the sum of the amount of storage used by all such pointers.

If a table has been compressed with **myisampack**, **myisamchk -d** prints additional information about each table column. See [Section 7.7, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#), for an example of this information and a description of what it means.

5.9.4.6. Setting Up a Table Maintenance Schedule

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. One way to check and repair **MyISAM** tables is with the **CHECK TABLE** and **REPAIR TABLE** statements. See [Section 12.5.2.3, “CHECK TABLE Syntax”](#), and [Section 12.5.2.6, “REPAIR TABLE Syntax”](#).

Another way to check tables is to use **myisamchk**. For maintenance purposes, you can use **myisamchk -s**. The **-s** option (short for **--silent**) causes **myisamchk** to run in silent mode, printing messages only when errors occur.

It is also a good idea to enable automatic **MyISAM** table checking. For example, whenever the machine has done a restart in the middle of an update, you usually need to check each table that could have been affected before it is used further. (These are “expected crashed tables.”) To check **MyISAM** tables automatically, start the server with the **--myisam-recover** option. See [Section 5.2.2, “Command Options”](#).

You should also check your tables regularly during normal system operation. At MySQL AB, we run a **cron** job to check all our important tables once a week, using a line like this in a **crontab** file:

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

This prints out information about crashed tables so that we can examine and repair them when needed.

Because we have not had any unexpectedly crashed tables (tables that become corrupted for reasons other than hardware trouble) for several years, once a week is more than sufficient for us.

We recommend that to start with, you execute **myisamchk -s** each night on all tables that have been updated during the last 24 hours, until you come to trust MySQL as much as we do.

Normally, MySQL tables need little maintenance. If you are performing many updates to **MyISAM** tables with dynamic-sized rows (tables with **VARCHAR**, **BLOB**, or **TEXT** columns) or have tables with many deleted rows you may want to defragment/reclaim space from the tables from time to time. You can do this by using **OPTIMIZE TABLE** on the tables in question. Alternatively, if you can stop the **mysqld** server for a while, change location into the data directory and use this command while the server is stopped:

```
shell> myisamchk -r -s --sort-index --sort_buffer_size=16M */*.MYI
```

5.10. MySQL Localization and International Usage

This section describes how to configure the server to use different character sets. It also discusses how to set the server's time zone and enable per-connection time zone support.

5.10.1. The Character Set Used for Data and Sorting

By default, MySQL uses the **latin1** (cp1252 West European) character set and the **latin1_swedish_ci** collation that sorts according to Swedish/Finnish rules. These defaults are suitable for the United States and most of Western Europe.

All MySQL binary distributions are compiled with **--with-extra-charsets=complex**. This adds code to all standard programs that enables them to handle **latin1** and all multi-byte character sets within the binary. Other character sets are loaded from a character-set definition file when needed.

The character set determines what characters are allowed in identifiers. The collation determines how strings are sorted by the **ORDER**

BY and GROUP BY clauses of the SELECT statement.

You can change the default server character set and collation with the `--character-set-server` and `--collation-server` options when you start the server. The collation must be a legal collation for the default character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.) See [Section 5.2.2, “Command Options”](#).

The character sets available depend on the `--with-charset=charset_name` and `--with-extra-charsets=list-of-charsets | complex | all | none` options to `configure`, and the character set configuration files listed in `SHAREDIR/charsets/Index`. See [Section 2.9.2, “Typical configure Options”](#).

If you change the character set when running MySQL, that may also change the sort order. Consequently, you must run `myisamchk -r -q --set-collation=collation_name` on all MyISAM tables, or your indexes may not be ordered correctly.

When a client connects to a MySQL server, the server indicates to the client what the server's default character set is. The client switches to this character set for this connection.

You should use `mysql_real_escape_string()` when escaping strings for an SQL query. `mysql_real_escape_string()` is identical to the old `mysql_escape_string()` function, except that it takes the MySQL connection handle as the first parameter so that the appropriate character set can be taken into account when escaping characters.

If the client is compiled with paths that differ from where the server is installed and the user who configured MySQL didn't include all character sets in the MySQL binary, you must tell the client where it can find the additional character sets it needs if the server runs with a different character set from the client. You can do this by specifying a `--character-sets-dir` option to indicate the path to the directory in which the dynamic MySQL character sets are stored. For example, you can put the following in an option file:

```
[client]
character-sets-dir=/usr/local/mysql/share/mysql/charsets
```

You can force the client to use specific character set as follows:

```
[client]
default-character-set=charset_name
```

This is normally unnecessary, however.

5.10.1.1. Using the German Character Set

In MySQL 5.1, character set and collation are specified separately. This means that if you want German sort order, you should select the `latin1` character set and either the `latin1_german1_ci` or `latin1_german2_ci` collation. For example, to start the server with the `latin1_german1_ci` collation, use the `--character-set-server=latin1` and `--collation-server=latin1_german1_ci` options.

For information on the differences between these two collations, see [Section 9.11.2, “West European Character Sets”](#).

5.10.2. Setting the Error Message Language

By default, `mysqld` produces error messages in English, but they can also be displayed in any of these other languages: Czech, Danish, Dutch, Estonian, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Norwegian-ny, Polish, Portuguese, Romanian, Russian, Slovak, Spanish, or Swedish.

To start `mysqld` with a particular language for error messages, use the `--language` or `-L` option. The option value can be a language name or the full path to the error message file. For example:

```
shell> mysqld --language=swedish
```

Or:

```
shell> mysqld --language=/usr/local/share/swedish
```

The language name should be specified in lowercase.

By default, the language files are located in the `share/LANGUAGE` directory under the MySQL base directory.

You can also change the content of the error messages produced by the server. Details can be found in the MySQL Internals manual, available at http://forge.mysql.com/wiki/MySQL_Internals_Error_Messages. If you upgrade to a newer version of MySQL after chan-

ging the error messages, remember to repeat your changes after the upgrade.

5.10.3. Adding a New Character Set

This section discusses the procedure for adding a new character set to MySQL. You must have a MySQL source distribution to use these instructions. To choose the proper procedure, determine whether the character set is simple or complex:

- If the character set does not need to use special string collating routines for sorting and does not need multi-byte character support, it is simple.
- If it needs either of those features, it is complex.

For example, `latin1` and `danish` are simple character sets, whereas `big5` and `czech` are complex character sets.

In the following instructions, the name of the character set is represented by `MYSET`.

For a simple character set, do the following:

1. Add `MYSET` to the end of the `sql/share/charsets/Index` file. Assign a unique number to it.
2. Create the file `sql/share/charsets/MYSET.conf`. (You can use a copy of `sql/share/charsets/latin1.conf` as the basis for this file.)

The syntax for the file is very simple:

- Comments start with a `#` character and continue to the end of the line.
- Words are separated by arbitrary amounts of whitespace.
- When defining the character set, every word must be a number in hexadecimal format.
- The `ctype` array takes up the first 257 words. The `to_lower[]`, `to_upper[]` and `sort_order[]` arrays take up 256 words each after that.

See [Section 5.10.4, “The Character Definition Arrays”](#).

3. Add the character set name to the `CHARSETS_AVAILABLE` and `COMPILED_CHARSETS` lists in `configure.in`.
4. Reconfigure, recompile, and test.

For a complex character set, do the following:

1. Create the file `strings/ctype-MYSET.c` in the MySQL source distribution.
2. Add `MYSET` to the end of the `sql/share/charsets/Index` file. Assign a unique number to it.
3. Look at one of the existing `ctype-*.c` files (such as `strings/ctype-big5.c`) to see what needs to be defined. Note that the arrays in your file must have names like `ctype_MYSET`, `to_lower_MYSET`, and so on. These correspond to the arrays for a simple character set. See [Section 5.10.4, “The Character Definition Arrays”](#).
4. Near the top of the file, place a special comment like this:

```
/*
 * This comment is parsed by configure to create ctype.c,
 * so don't change it unless you know what you are doing.
 *
 * .configure. number_MYSET=MYNUMBER
 * .configure. strxfrm_multiply_MYSET=N
 * .configure. mbmaxlen_MYSET=N
 */
```

The `configure` program uses this comment to include the character set into the MySQL library automatically.

The `strxfrm_multiply` and `mbmaxlen` lines are explained in the following sections. You need include them only if you need

the string collating functions or the multi-byte character set functions, respectively.

5. You should then create some of the following functions:

- `my_strncoll_MYSET()`
- `my_strcoll_MYSET()`
- `my_strxfrm_MYSET()`
- `my_like_range_MYSET()`

See [Section 5.10.5, “String Collating Support”](#).

6. Add the character set name to the `CHARSETS_AVAILABLE` and `COMPILED_CHARSETS` lists in `configure.in`.

7. Reconfigure, recompile, and test.

The `sql/share/charsets/README` file includes additional instructions.

If you want to have the character set included in the MySQL distribution, mail a patch to the MySQL [internals](#) mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#).

5.10.4. The Character Definition Arrays

`to_lower[]` and `to_upper[]` are simple arrays that hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

```
to_lower['A'] should contain 'a'
to_upper['a'] should contain 'A'
```

`sort_order[]` is a map indicating how characters should be ordered for comparison and sorting purposes. Quite often (but not for all character sets) this is the same as `to_upper[]`, which means that sorting is case-insensitive. MySQL sorts characters based on the values of `sort_order[]` elements. For more complicated sorting rules, see the discussion of string collating in [Section 5.10.5, “String Collating Support”](#).

`ctype[]` is an array of bit values, with one element for one character. (Note that `to_lower[]`, `to_upper[]`, and `sort_order[]` are indexed by character value, but `ctype[]` is indexed by character value + 1. This is an old legacy convention for handling EOF.)

You can find the following bitmask definitions in `m_ctype.h`:

```
#define _U      01      /* Uppercase */
#define _L      02      /* Lowercase */
#define _N      04      /* Numeral (digit) */
#define _S      010     /* Spacing character */
#define _P      020     /* Punctuation */
#define _C      040     /* Control character */
#define _B      0100    /* Blank */
#define _X      0200    /* hexadecimal digit */
```

The `ctype[]` entry for each character should be the union of the applicable bitmask values that describe the character. For example, 'A' is an uppercase character (`_U`) as well as a hexadecimal digit (`_X`), so `ctype['A'+1]` should contain the value:

```
_U + _X = 01 + 0200 = 0201
```

5.10.5. String Collating Support

If the sorting rules for your language are too complex to be handled with the simple `sort_order[]` table, you need to use the string collating functions.

The best documentation for this is the existing character sets. Look at the [big5](#), [czech](#), [gbk](#), [sjis](#), and [tis160](#) character sets for examples.

You must specify the `strxfrm_multiply_MYSET=N` value in the special comment at the top of the file. *N* should be set to the max-

imum ratio the strings may grow during `my_strxfrm_MYSET` (it must be a positive integer).

5.10.6. Multi-Byte Character Support

If you want to add support for a new character set that includes multi-byte characters, you need to use the multi-byte character functions.

The best documentation for this is the existing character sets. Look at the `euc_kr`, `gb2312`, `gbk`, `sjis`, and `ujis` character sets for examples. These are implemented in the `ctype-charset_name.c` files in the `strings` directory.

You must specify the `mbmaxlen_MYSET=N` value in the special comment at the top of the source file. *N* should be set to the size in bytes of the largest character in the set.

5.10.7. Problems With Character Sets

If you try to use a character set that is not compiled into your binary, you might run into the following problems:

- Your program uses an incorrect path to determine where the character sets are stored. (Default `/usr/local/mysql/share/mysql/charsets`). This can be fixed by using the `--character-sets-dir` option when you run the program in question.
- The character set is a multi-byte character set that cannot be loaded dynamically. In this case, you must recompile the program with support for the character set.
- The character set is a dynamic character set, but you do not have a configure file for it. In this case, you should install the configure file for the character set from a new MySQL distribution.
- If your `Index` file does not contain the name for the character set, your program displays the following error message:

```
ERROR 1105: File '/usr/local/share/mysql/charsets/?.conf'
not found (Errcode: 2)
```

In this case, you should either get a new `Index` file or manually add the name of any missing character sets to the current file.

For `MyISAM` tables, you can check the character set name and number for a table with `myisamchk -dvv tbl_name`.

5.10.8. MySQL Server Time Zone Support

The MySQL server maintains several time zone settings:

- The system time zone. When the server starts, it attempts to determine the time zone of the host machine and uses it to set the `system_time_zone` system variable. The value does not change thereafter.

You can set the system time zone for MySQL Server at startup with the `--timezone=timezone_name` option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`. The allowable values for `--timezone` or `TZ` are system-dependent. Consult your operating system documentation to see what values are acceptable.

- The server's current time zone. The global `time_zone` system variable indicates the time zone the server currently is operating in. The initial value for `time_zone` is `'SYSTEM'`, which indicates that the server time zone is the same as the system time zone.

The initial global server time zone value can be specified explicitly at startup with the `--default-time-zone=timezone` option on the command line, or you can use the following line in an option file:

```
default-time-zone='timezone'
```

If you have the `SUPER` privilege, you can set the global server time zone value at runtime with this statement:

```
mysql> SET GLOBAL time_zone = timezone;
```

- Per-connection time zones. Each client that connects has its own time zone setting, given by the session `time_zone` variable. Initially, the session variable takes its value from the global `time_zone` variable, but the client can change its own time zone with

this statement:

```
mysql> SET time_zone = timezone;
```

The current session time zone setting affects display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` or `CURTIME()`, and values stored in and retrieved from `TIMESTAMP` columns. Values for `TIMESTAMP` columns are converted from the current time zone to UTC for storage, and from UTC to the current time zone for retrieval. The current time zone setting does not affect values displayed by functions such as `UTC_TIMESTAMP()` or values in `DATE`, `TIME`, or `DATETIME` columns.

The current values of the global and client-specific time zones can be retrieved like this:

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
```

`timezone` values can be given in several formats, none of which are case sensitive:

- The value `'SYSTEM'` indicates that the time zone should be the same as the system time zone.
- The value can be given as a string indicating an offset from UTC, such as `'+10:00'` or `'-6:00'`.
- The value can be given as a named time zone, such as `'Europe/Helsinki'`, `'US/Eastern'`, or `'MET'`. Named time zones can be used only if the time zone information tables in the `mysql` database have been created and populated.

The MySQL installation procedure creates the time zone tables in the `mysql` database, but does not load them. You must do so manually using the following instructions. (If you are upgrading to MySQL 4.1.3 or later from an earlier version, you can create the tables by upgrading your `mysql` database. Use the instructions in [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#). After creating the tables, you can load them.)

Note

Loading the time zone information is not necessarily a one-time operation because the information changes occasionally. For example, the rules for Daylight Saving Time in the United States, Mexico, and parts of Canada changed in 2007. When such changes occur, applications that use the old rules become out of date and you may find it necessary to reload the time zone tables to keep the information used by your MySQL server current. See the notes at the end of this section.

If your system has its own `zoneinfo` database (the set of files describing time zones), you should use the `mysql_tzinfo_to_sql` program for filling the time zone tables. Examples of such systems are Linux, FreeBSD, Sun Solaris, and Mac OS X. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system does not have a `zoneinfo` database, you can use the downloadable package described later in this section.

The `mysql_tzinfo_to_sql` program is used to load the time zone tables. On the command line, pass the `zoneinfo` directory path-name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

`mysql_tzinfo_to_sql` also can be used to load a single time zone file or to generate leap second information:

- To load a single time zone file `tz_file` that corresponds to a time zone name `tz_name`, invoke `mysql_tzinfo_to_sql` like this:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

With this approach, you must execute a separate command to load the time zone file for each named zone that the server needs to know about.

- If your time zone needs to account for leap seconds, initialize the leap second information like this, where `tz_file` is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

If your system is one that has no zoneinfo database (for example, Windows or HP-UX), you can use the package of pre-built time zone tables that is available for download at the MySQL Developer Zone:

<http://dev.mysql.com/downloads/timezones.html>

This time zone package contains `.frm`, `.MYD`, and `.MYI` files for the `MyISAM` time zone tables. These tables should be part of the `mysql` database, so you should place the files in the `mysql` subdirectory of your MySQL server's data directory. The server should be stopped while you do this and restarted afterward.

Warning

Do not use the downloadable package if your system has a zoneinfo database. Use the `mysql_tzinfo_to_sql` utility instead. Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

For information about time zone settings in replication setup, please see [Section 15.3.1, “Replication Features and Issues”](#).

Staying Current with Time Zone Changes

As mentioned earlier, when the time zone rules change, applications that use the old rules become out of date. To stay current, it is necessary to make sure that your system uses current time zone information is used. For MySQL, there are two factors to consider in staying current:

- The operating system time affects the value that the MySQL server uses for times if its time zone is set to `SYSTEM`. Make sure that your operating system is using the latest time zone information. For most operating systems, the latest update or service pack prepares your system for the time changes. Check the Web site for your operating system vendor for an update that addresses the time changes.
- If you replace the system's `/etc/localtime` timezone file with a version that uses rules differing from those in effect at `mysqld` startup, you should restart `mysqld` so that it uses the updated rules. Otherwise, `mysqld` might not notice when the system changes its time.
- If you use named time zones with MySQL, make sure that the time zone tables in the `mysql` database are up to date. If your system has its own zoneinfo database, you should reload the MySQL time zone tables whenever the zoneinfo database is updated, using the instructions given earlier in this section. For systems that do not have their own zoneinfo database, check the MySQL Developer Zone for updates. When a new update is available, download it and use it to replace your current time zone tables. `mysqld` caches time zone information that it looks up, so after replacing the time zone tables, you should restart `mysqld` to make sure that it does not continue to serve outdated time zone data.

If you are uncertain whether named time zones are available, for use either as the server's time zone setting or by clients that set their own time zone, check whether your time zone tables are empty. The following query determines whether the table that contains time zone names has any rows:

```
mysql> SELECT COUNT(*) FROM mysql.time_zone_name;
+-----+
| COUNT(*) |
+-----+
|         0 |
+-----+
```

A count of zero indicates that the table is empty. In this case, no one can be using named time zones, and you don't need to update the tables. A count greater than zero indicates that the table is not empty and that its contents are available to be used for named time zone support. In this case, you should be sure to reload your time zone tables so that anyone who uses named time zones will get correct query results.

To check whether your MySQL installation is updated properly for a change in Daylight Saving Time rules, use a test like the one following. The example uses values that are appropriate for the 2007 DST 1-hour change that occurs in the United States on March 11 at 2 a.m.

The test uses these two queries:

```
SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
```

The two time values indicate the times at which the DST change occurs, and the use of named time zones requires that the time zone tables be used. The desired result is that both queries return the same result (the input time, converted to the equivalent value in the 'US/Central' time zone).

Before updating the time zone tables, you would see an incorrect result like this:

```
mysql> SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
+-----+
| CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') |
+-----+
| 2007-03-11 01:00:00 |
+-----+

mysql> SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
+-----+
| CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') |
+-----+
| 2007-03-11 02:00:00 |
+-----+
```

After updating the tables, you should see the correct result:

```
mysql> SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
+-----+
| CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') |
+-----+
| 2007-03-11 01:00:00 |
+-----+

mysql> SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
+-----+
| CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') |
+-----+
| 2007-03-11 01:00:00 |
+-----+
```

5.10.9. MySQL Server Locale Support

Beginning with MySQL 5.1.12, the locale indicated by the `lc_time_names` system variable controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()` and `MONTHNAME()` functions.

Locale names are POSIX-style values such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting, but any client can examine or set its `lc_time_names` value as shown in the following example:

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @@lc_time_names;
+-----+
| @@lc_time_names |
+-----+
| en_US |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----+-----+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----+-----+
| Friday | January |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----+
| Friday Fri January Jan |
+-----+
1 row in set (0.00 sec)

mysql> SET lc_time_names = 'es_MX';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@lc_time_names;
```

```

+-----+
| @@lc_time_names |
+-----+
| es_MX           |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----+-----+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----+-----+
| viernes              | enero                   |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----+
| viernes vie enero ene                   |
+-----+
1 row in set (0.00 sec)

```

The day or month name for each of the affected functions is converted from `utf8` to the character set indicated by the `character_set_connection` system variable.

`lc_time_names` may be set to any of the following locale values:

<code>ar_AE</code> : Arabic - United Arab Emirates	<code>ar_BH</code> : Arabic - Bahrain
<code>ar_DZ</code> : Arabic - Algeria	<code>ar_EG</code> : Arabic - Egypt
<code>ar_IN</code> : Arabic - Iran	<code>ar_IQ</code> : Arabic - Iraq
<code>ar_JO</code> : Arabic - Jordan	<code>ar_KW</code> : Arabic - Kuwait
<code>ar_LB</code> : Arabic - Lebanon	<code>ar_LY</code> : Arabic - Libya
<code>ar_MA</code> : Arabic - Morocco	<code>ar_OM</code> : Arabic - Oman
<code>ar_QA</code> : Arabic - Qatar	<code>ar_SA</code> : Arabic - Saudi Arabia
<code>ar_SD</code> : Arabic - Sudan	<code>ar_SY</code> : Arabic - Syria
<code>ar_TN</code> : Arabic - Tunisia	<code>ar_YE</code> : Arabic - Yemen
<code>be_BY</code> : Belarusian - Belarus	<code>bg_BG</code> : Bulgarian - Bulgaria
<code>ca_ES</code> : Catalan - Catalan	<code>cs_CZ</code> : Czech - Czech Republic
<code>da_DK</code> : Danish - Denmark	<code>de_AT</code> : German - Austria
<code>de_BE</code> : German - Belgium	<code>de_CH</code> : German - Switzerland
<code>de_DE</code> : German - Germany	<code>de_LU</code> : German - Luxembourg
<code>EE</code> : Estonian - Estonia	<code>en_AU</code> : English - Australia
<code>en_CA</code> : English - Canada	<code>en_GB</code> : English - United Kingdom
<code>en_IN</code> : English - India	<code>en_NZ</code> : English - New Zealand
<code>en_PH</code> : English - Philippines	<code>en_US</code> : English - United States
<code>en_ZA</code> : English - South Africa	<code>en_ZW</code> : English - Zimbabwe
<code>es_AR</code> : Spanish - Argentina	<code>es_BO</code> : Spanish - Bolivia
<code>es_CL</code> : Spanish - Chile	<code>es_CO</code> : Spanish - Columbia
<code>es_CR</code> : Spanish - Costa Rica	<code>es_DO</code> : Spanish - Dominican Republic
<code>es_EC</code> : Spanish - Ecuador	<code>es_ES</code> : Spanish - Spain
<code>es_GT</code> : Spanish - Guatemala	<code>es_HN</code> : Spanish - Honduras
<code>es_MX</code> : Spanish - Mexico	<code>es_NI</code> : Spanish - Nicaragua
<code>es_PA</code> : Spanish - Panama	<code>es_PE</code> : Spanish - Peru
<code>es_PR</code> : Spanish - Puerto Rico	<code>es_PY</code> : Spanish - Paraguay
<code>es_SV</code> : Spanish - El Salvador	<code>es_US</code> : Spanish - United States
<code>es_UY</code> : Spanish - Uruguay	<code>es_VE</code> : Spanish - Venezuela
<code>eu_ES</code> : Basque - Basque	<code>fi_FI</code> : Finnish - Finland

fo_FO : Faroese - Faroe Islands	fr_BE : French - Belgium
fr_CA : French - Canada	fr_CH : French - Switzerland
fr_FR : French - France	fr_LU : French - Luxembourg
gl_ES : Galician - Galician	gu_IN : Gujarati - India
he_IL : Hebrew - Israel	hi_IN : Hindi - India
hr_HR : Croatian - Croatia	hu_HU : Hungarian - Hungary
id_ID : Indonesian - Indonesia	is_IS : Icelandic - Iceland
it_CH : Italian - Switzerland	it_IT : Italian - Italy
ja_JP : Japanese - Japan	ko_KR : Korean - Korea
lt_LT : Lithuanian - Lithuania	lv_LV : Latvian - Latvia
mk_MK : Macedonian - FYROM	mn_MN : Mongolia - Mongolian
ms_MY : Malay - Malaysia	nb_NO : Norwegian(Bokml) - Norway
nl_BE : Dutch - Belgium	nl_NL : Dutch - The Netherlands
no_NO : Norwegian - Norway	pl_PL : Polish - Poland
pt_BR : Portugese - Brazil	pt_PT : Portugese - Portugal
ro_RO : Romanian - Romania	ru_RU : Russian - Russia
ru-UA : Russian - Ukraine	sk_SK : Slovak - Slovakia
sl_SI : Slovenian - Slovenia	sq_AL : Albanian - Albania
sr_YU : Serbian - Yugoslavia	sv_FI : Swedish - Finland
sv_SE : Swedish - Sweden	ta_IN : Tamil - India
te_IN : Telugu - India	th_TH : Thai - Thailand
tr_TR : Turkish - Turkey	uk-UA : Ukrainian - Ukraine
ur_PK : Urdu - Pakistan	vi_VN : Vietnamese - Vietnam
zh_CN : Chinese - Peoples Republic of China	zh_HK : Chinese - Hong Kong SAR
zh_TW : Chinese - Taiwan	

[lc_time_names](#) currently does not affect the [STR_TO_DATE\(\)](#) or [GET_FORMAT\(\)](#) function.

5.11. MySQL Server Logs

MySQL has several different logs that can help you find out what is going on inside [mysqld](#):

Log Type	Information Written to Log
The error log	Problems encountered starting, running, or stopping mysqld
The general query log	Established client connections and statements received from clients
The binary log	All statements that change data (also used for replication)
The slow query log	All queries that took more than long_query_time seconds to execute or didn't use indexes

By default, all log files are created in the [mysqld](#) data directory. You can force [mysqld](#) to close and reopen the log files (or in some cases switch to a new log) by flushing the logs. Log flushing occurs when you issue a [FLUSH LOGS](#) statement or execute [mysqldadmin flush-logs](#) or [mysqldadmin refresh](#). See [Section 12.5.5.2, “FLUSH Syntax”](#), and [Section 7.10, “mysqldadmin — Client for Administering a MySQL Server”](#).

If you are using MySQL replication capabilities, slave replication servers maintain additional log files called relay logs. [Chapter 15, Replication](#), discusses relay log contents and configuration.

MySQL Enterprise

The MySQL Enterprise Monitor provides a number of advisors specifically related to the various log files. For

more information see <http://www.mysql.com/products/enterprise/advisors.html>.

As of MySQL 5.1.6, the server can write general query and slow query entries to log tables, log files, or both. For details, see [Section 5.11.1, “Selecting General Query and Slow Query Log Output Destinations”](#).

As of MySQL 5.1.12, additional runtime control of the general query and slow query logs is available: You can enable or disable logging, or change the name of the log file. See [Section 5.11.3, “The General Query Log”](#), and [Section 5.11.5, “The Slow Query Log”](#).

5.11.1. Selecting General Query and Slow Query Log Output Destinations

As of MySQL 5.1.6, the server provides flexible control over the destination for log output. Log entries can be written to log files to the `general_log` and `slow_log` tables in the `mysql` database. If logging is enabled, either or both destinations can be selected. (Before MySQL 5.1.6, the server uses only log files as the destination for general query log and slow query log entries, if those logs are enabled.)

Note

For new installations of MySQL 5.1.6 or higher, the log tables are created during the installation procedure along with the other system tables. If you upgrade MySQL from a release older than 5.1.6 to MySQL 5.1.6 or higher, you must upgrade the system tables after upgrading to make sure that the log tables exist. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

Currently, logging to tables incurs significantly more server overhead than logging to files. If you enable the general log or slow query log and require highest performance, you should log to files and not to tables.

The `--log-output` option specifies the destination for log output, if logging is enabled, but the option does not in itself enable the logs. The syntax for this option is `--log-output[=value, ...]`:

- If `--log-output` is given with a value, the value can be a comma-separated list of one or more of the words `TABLE` (log to tables), `FILE` (log to files), or `NONE` (do not log to tables or files). `NONE`, if present, takes precedence over any other specifiers.
- If `--log-output` is omitted or given without a value, the effect is the same as `--log-output=FILE`. That is, the file destination is selected. (For MySQL 5.1.6 through 5.1.20, the default is the table destination.)
- If `--log-output` option also sets the value of the global `log_output` system variable, which can be modified at runtime to change the logging destination for the server while it executes.

The `--log[=file_name]` option, if given, enables logging to the general query log for the selected log destinations. Similarly, the `--log-slow-queries[=file_name]` option, if given, enables logging to the slow query log for the selected destinations. If you specify either option, the server opens the corresponding log file and writes startup messages to it. However, logging of queries to the file does not occur unless the `FILE` log destination is selected.

Examples:

- To write general query log entries to the log table and the log file, use `--log-output=TABLE, FILE` to select both log destinations and the `--log` option to enable the general query log.
- To write general and slow query log entries only to the log tables, use `--log-output=TABLE` to select tables as the log destination and the `--log` and `--log-slow-queries` options to enable both logs. (In this case, because the default log destination is `TABLE`, you could omit the `--log-output` option.)

Several system variables are associated with log tables and files:

- The global `log` and `slow_query_log` variables indicate whether the general query log and slow query log are enabled (`ON`) or disabled (`OFF`). You can set these variables at runtime to control whether the logs are enabled.
- The global `general_log_file` and `slow_query_log_file` variables indicate the names of the general query log and slow query log files. You can set these variables at runtime to change the names of the log files. (If the `--log` and `--log-slow-queries` options were not given, the variable values are set to the default log filenames.)
- The session `sql_log_off` variable can be set to `ON` or `OFF` to disable or enable general query logging for the current connection.

The use of tables for log output offers the following benefits:

- Log entries have a standard format. To display the current structure of the log tables, use these statements:

```
SHOW CREATE TABLE mysql.general_log;  
SHOW CREATE TABLE mysql.slow_log;
```

- Log contents are accessible via SQL statements. This enables the use of queries that select only those log entries that satisfy specific criteria. For example, to select log contents associated with a particular client (which can be useful for identifying problematic queries from that client), it is easier to do this using a log table than a log file.
- Logs are accessible remotely through any client that can connect to the server and issue queries (if the client has the appropriate log table privileges). It's not necessary to log in to the server host and directly access the filesystem.

The log table implementation has the following characteristics:

- In general, the primary purpose of log tables is to provide an interface for users to observe the runtime execution of the server, not to interfere with its runtime execution.
- **CREATE TABLE**, **ALTER TABLE**, and **DROP TABLE** are valid operations on a log table. For **ALTER TABLE** and **DROP TABLE**, the log table cannot be in use and must be disabled, as described later.
- By default, the log tables use the **CSV** storage engine that writes data in comma-separated values format. For users who have access to the **.CSV** files that contain log table data, the files are easy to import into other programs such as spreadsheets that can process CSV input.

Beginning with MySQL 5.1.12, the log tables can be altered to use the **MyISAM** storage engine. You cannot use **ALTER TABLE** to alter a log table that is in use. The log must be disabled first. No engines other than **CSV** or **MyISAM** are legal for the log tables.

- To disable logging so that you can alter (or drop) a log table, you can use the following strategy. The example uses the general query log; the procedure for the slow query log is similar but uses the **slow_log** table and **slow_query_log** system variable.

```
SET @old_log_state = @@global.general_log;  
SET GLOBAL general_log = 'OFF';  
ALTER TABLE mysql.general_log ENGINE = MyISAM;  
SET GLOBAL general_log = @old_log_state;
```

- **TRUNCATE TABLE** is a valid operation on a log table. It can be used to expire log entries.
- As of MySQL 5.1.13, **RENAME TABLE** is a valid operation on a log table. You can atomically rename a log table (to perform log rotation, for example) using the following strategy:

```
USE mysql;  
CREATE TABLE IF NOT EXISTS general_log2 LIKE general_log;  
RENAME TABLE general_log TO general_log_backup, general_log2 TO general_log;
```

- **LOCK TABLES** cannot be used on a log table.
- **INSERT**, **DELETE**, and **UPDATE** cannot be used on a log table. These operations are allowed only internally to the server itself.
- The global read lock and the state of the global **read_only** system variable have no effect on log tables. The server can always write to the log tables.
- Entries written to the log tables are not written to the binary log and thus are not replicated to slave servers.
- To flush the log tables or log files, use **FLUSH TABLES** or **FLUSH LOGS**, respectively. (From MySQL 5.1.12 to 5.1.20, **FLUSH TABLES** ignores log tables and **FLUSH LOGS** flushes both the log tables and files.)
- It is not recommended to partition log tables, and doing so is not allowed beginning with MySQL 5.1.20.

5.11.2. The Error Log

The error log contains information indicating when `mysqld` was started and stopped and also any critical errors that occur while the server is running. If `mysqld` notices a table that needs to be automatically checked or repaired, it writes a message to the error log.

On some operating systems, the error log contains a stack trace if `mysqld` dies. The trace can be used to determine where `mysqld` died. See [MySQL Internals: Porting](#).

You can specify where `mysqld` writes the error log with the `--log-error[=file_name]` option. If no `file_name` value is given, `mysqld` uses the name `host_name.err` by default and writes the file in the data directory. If you execute `FLUSH LOGS`, the error log is renamed with the suffix `-old` and `mysqld` creates a new empty log file. (No renaming occurs if the `--log-error` option was not given.)

If you do not specify `--log-error`, or (on Windows) if you use the `--console` option, errors are written to `stderr`, the standard error output. Usually this is your terminal.

On Windows, error output is always written to the `.err` file if `--console` is not given.

The `--log-warnings` option or `log_warnings` system variable can be used to control warning logging to the error log. The default value is enabled (1). Warning logging can be disabled using a value of 0. Aborted connections are not logged to the error log unless the value is greater than 1. See [Section B.1.2.10, “Communication Errors and Aborted Connections”](#).

If you use `mysqld_safe` to start `mysqld`, `mysqld_safe` arranges for `mysqld` to write error messages to a log file or (as of MySQL 5.1.20) to `syslog`:

- Before 5.1.20, `mysqld_safe` behavior is to log to a file, using the default error log file if the `--log-error` option is not given to `mysqld_safe`. Otherwise, `mysqld_safe` uses the filename specified via `--log-error=file_name`.
- From 5.1.20 on, `mysqld_safe` has two additional error-logging options, `--syslog` and `--skip-syslog`.

In 5.1.21 and up, the default with no logging options is `--skip-syslog`, which is compatible with the default behavior of writing an error log file for releases prior to 5.1.20. To explicitly specify use of an error log file, specify `--log-error=file_name` to `mysqld_safe`, and `mysqld_safe` will arrange for `mysqld` to write messages to a log file. To use `syslog` instead, specify the `--syslog` option.

In 5.1.20 only, the following conditions apply: 1) The default is to use `syslog`, which is not compatible with releases prior to 5.1.20. 2) Logging to `syslog` may fail to operate correctly in some cases, so we recommend that you use `--skip-syslog` or `--log-error`.

For logging to `syslog`, messages from `mysqld_safe` and `mysqld` are written with a tag of `mysqld_safe` and `mysqld`, respectively. As of MySQL 5.1.21, to specify a suffix for the tag, use `--syslog-tag=tag`, which modifies the tags to be `mysqld_safe-tag` and `mysqld-tag`.

If you specify `--log-error` in an option file in a section that `mysqld` reads, `mysqld_safe` also will find and use the option.

If `mysqld_safe` is used to start `mysqld` and `mysqld` dies unexpectedly, `mysqld_safe` notices that it needs to restart `mysqld` and writes a `restarted mysqld` message to the error log.

5.11.3. The General Query Log

The general query log is a general record of what `mysqld` is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to `mysqld`.

`mysqld` writes statements to the query log in the order that it receives them, which might differ from the order in which they are executed. This logging order contrasts to the binary log, for which statements are written after they are executed but before any locks are released. (Also, the query log contains all statements, whereas the binary log does not contain statements that only select data.)

To enable the general query log as of MySQL 5.1.6, start `mysqld` with the `--log[=file_name]` or `-l [file_name]` option, and optionally use `--log-output` to specify the log destination (as described in [Section 5.11.1, “Selecting General Query and Slow Query Log Output Destinations”](#)). Before 5.1.6, the general query log destination is always a file. To enable the general query log file, use the `--log[=file_name]` or `-l [file_name]` option.

If no `file_name` value is given for `--log` or `-l`, the default name is `host_name.log` in the data directory. If a filename is given, but not as an absolute pathname, the server writes the file in the data directory.

When `--log` or `-l` is specified, the `--general-log` option also may be given as of MySQL 5.1.12 to specify the initial general query log state. With no argument or an argument of 0, the option disables the log. If omitted or given with an argument of 1, the option enables the log. If `--log` or `-l` is not specified, `--general-log` has no effect.

The global `general_log` and `general_log_file` system variables provide runtime control over the general query log. Set `general_log` to 0 (or `OFF`) to disable the log or to 1 (or `ON`) to enable it. Set `general_log_file` to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

When the general query log is enabled, output is written to any destinations specified by the `--log-output` option or `log_output` system variable. Note that if the destination is `NONE`, no output is written even if the general log is enabled. Setting the log filename has no effect on logging if the log destination value does not contain `FILE`.

Server restarts and log flushing do not cause a new general query log file to be generated (although flushing closes and reopens it). On Unix, you can rename the file and create a new one by using the following commands:

```
shell> mv host_name.log host_name-old.log
shell> mysqladmin flush-logs
shell> cp host_name-old.log backup-directory
shell> rm host_name-old.log
```

On Windows, you cannot rename the log file while the server has it open. You must stop the server and rename the file, and then restart the server to create a new log file.

As of MySQL 5.1.12, you can disable the general query log at runtime:

```
SET GLOBAL general_log = 'OFF';
```

With the log disabled, rename the log file externally; for example, from the command line. Then enable the log again:

```
SET GLOBAL general_log = 'ON';
```

This method works on any platform and does not require a server restart.

The session `sql_log_off` variable can be set to `ON` or `OFF` to disable or enable general query logging for the current connection.

5.11.4. The Binary Log

The binary log contains all statements that update data or potentially could have updated it (for example, a `DELETE` which matched no rows). Statements are stored in the form of “events” that describe the modifications. The binary log also contains information about how long each statement took that updated data.

The binary log is not used for statements such as `SELECT` or `SHOW` that do not modify data. If you want to log all statements (for example, to identify a problem query), use the general query log. See [Section 5.11.3, “The General Query Log”](#).

The format of the events recorded in the binary log is dependent on the binary logging format. Three format types are supported, row-based logging, statement-based logging and mixed-base logging. The binary logging format used depends on the MySQL version. For more information on logging formats, see [Section 5.11.4.1, “Binary Logging Formats”](#).

The primary purpose of the binary log is to be able to update databases during a restore operation as fully as possible, because the binary log contains all updates done after a backup was made. The binary log is also used on master replication servers as a record of the statements to be sent to slave servers. See [Chapter 15, Replication](#).

MySQL Enterprise

The binary log can also be used to track significant DDL events. Analyzing the binary log in this way is an integral part of the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

Running the server with the binary log enabled makes performance about 1% slower. However, the benefits of the binary log for restore operations and in allowing you to set up replication generally outweigh this minor performance decrement.

When started with the `--log-bin[=base_name]` option, `mysqld` writes a log file containing all SQL commands that update data (both DDL and DML statements). If no `base_name` value is given, the default name is the value of the `pid-file` option (which by default is the name of host machine) followed by `-bin`. If the basename is given, but not as an absolute pathname, the server writes the file in the data directory. It is recommended that you specify a basename; see [Section B.1.8.1, “Open Issues in MySQL”](#), for the reason.

If you supply an extension in the log name (for example, `--log-bin=base_name.extension`), the extension is silently removed and ignored.

`mysqld` appends a numeric extension to the binary log basename. The number increases each time the server creates a new log file, thus creating an ordered series of files. The server creates a new binary log file each time it starts or flushes the logs. The server also creates a new binary log file automatically when the current log's size reaches `max_binlog_size`. A binary log file may become larger than `max_binlog_size` if you are using large transactions because a transaction is written to the file in one piece, never split between files.

To keep track of which binary log files have been used, `mysqld` also creates a binary log index file that contains the names of all used binary log files. By default this has the same basename as the binary log file, with the extension `'.index'`. You can change the name of the binary log index file with the `--log-bin-index[=file_name]` option. You should not manually edit this file while `mysqld` is running; doing so would confuse `mysqld`.

Replication slave servers by default do not write to their own binary log any statements that are received from the replication master. To cause these statements to be logged, start the slave with the `--log-slave-updates` option.

Writes to the binary log file and binary log index file are handled in the same way as writes to `MyISAM` tables. See [Section B.1.4.3, “How MySQL Handles a Full Disk”](#).

You can delete all binary log files with the `RESET MASTER` statement, or a subset of them with `PURGE MASTER LOGS`. See [Section 12.5.5.5, “RESET Syntax”](#), and [Section 12.6.1.1, “PURGE MASTER LOGS Syntax”](#).

The binary log format has some known limitations that can affect recovery from backups. See [Section 15.3.1, “Replication Features and Issues”](#).

Binary logging for stored routines and triggers is done as described in [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

You can use the following options to `mysqld` to affect what is logged to the binary log. See also the discussion that follows this option list.

If you are using replication, the options described here affect which statements are sent by a master server to its slaves. There are also options for slave servers that control which statements received from the master to execute or ignore. For details, see [Section 15.1.3, “Replication Options and Variables”](#).

- `--binlog-do-db=db_name`

Tell the server to restrict binary logging to updates for which the default database is `db_name` (that is, the database selected by `USE`). All other databases that are not explicitly mentioned are ignored. If you use this option, you should ensure that you do updates only in the default database.

There is an exception to this for `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. The server uses the database named in the statement (not the default database) to decide whether it should log the statement.

An example of what does not work as you might expect: If the server is started with `binlog-do-db=sales`, and you run `USE prices; UPDATE sales.january SET amount=amount+1000;`, this statement is *not* written into the binary log.

To log multiple databases, use multiple options, specifying the option once for each database.

- `--binlog-ignore-db=db_name`

Tell the server to suppress binary logging of updates for which the default database is `db_name` (that is, the database selected by `USE`). If you use this option, you should ensure that you do updates only in the default database.

As with the `--binlog-do-db` option, there is an exception for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. The server uses the database named in the statement (not the default database) to decide whether it should log the statement.

An example of what does not work as you might expect: If the server is started with `binlog-ignore-db=sales`, and you run `USE prices; UPDATE sales.january SET amount=amount+1000;`, this statement is written into the binary log.

To ignore multiple databases, use multiple options, specifying the option once for each database.

The server evaluates the options for logging or ignoring updates to the binary log according to the following rules. As described previ-

ously, there is an exception for the [CREATE DATABASE](#), [ALTER DATABASE](#), and [DROP DATABASE](#) statements. In those cases, the database being *created*, *altered*, or *dropped* replaces the default database in the following rules:

1. Are there [--binlog-do-db](#) or [--binlog-ignore-db](#) rules?
 - No: Write the statement to the binary log and exit.
 - Yes: Go to the next step.
2. There are some rules ([--binlog-do-db](#), [--binlog-ignore-db](#), or both). Is there a default database (has any database been selected by [USE](#))?
 - No: Do *not* write the statement, and exit.
 - Yes: Go to the next step.
3. There is a default database. Are there some [--binlog-do-db](#) rules?
 - Yes: Does the default database match any of the [--binlog-do-db](#) rules?
 - Yes: Write the statement and exit.
 - No: Do *not* write the statement, and exit.
 - No: Go to the next step.
4. There are some [--binlog-ignore-db](#) rules. Does the default database match any of the [--binlog-ignore-db](#) rules?
 - Yes: Do not write the statement, and exit.
 - No: Write the query and exit.

For example, a slave running with only [--binlog-do-db=sales](#) does not write to the binary log any statement for which the default database is different from [sales](#) (in other words, [--binlog-do-db](#) can sometimes mean “ignore other databases”).

If you are using replication, you should not delete old binary log files until you are sure that no slave still needs to use them. For example, if your slaves never run more than three days behind, once a day you can execute [mysqladmin flush-logs](#) on the master and then remove any logs that are more than three days old. You can remove the files manually, but it is preferable to use [PURGE MASTER LOGS](#), which also safely updates the binary log index file for you (and which can take a date argument). See [Section 12.6.1.1, “PURGE MASTER LOGS Syntax”](#).

A client that has the [SUPER](#) privilege can disable binary logging of its own statements by using a [SET SQL_LOG_BIN=0](#) statement. See [Section 12.5.3, “SET Syntax”](#).

You can display the contents of binary log files with the [mysqlbinlog](#) utility. This can be useful when you want to reprocess statements in the log. For example, you can update a MySQL server from the binary log as follows:

```
shell> mysqlbinlog log_file | mysql -h server_name
```

See [Section 7.11, “mysqlbinlog — Utility for Processing Binary Log Files”](#), for more information on the [mysqlbinlog](#) utility and how to use it. [mysqlbinlog](#) also can be used with relay log files because they are written using the same format as binary log files.

Binary logging is done immediately after a statement completes but before any locks are released or any commit is done. This ensures that the log is logged in execution order.

Updates to non-transactional tables are stored in the binary log immediately after execution. Within an uncommitted transaction, all updates ([UPDATE](#), [DELETE](#), or [INSERT](#)) that change transactional tables such as [InnoDB](#) tables are cached until a [COMMIT](#) statement is received by the server. At that point, [mysqld](#) writes the entire transaction to the binary log before the [COMMIT](#) is executed. When the thread that handles the transaction starts, it allocates a buffer of [binlog_cache_size](#) to buffer statements. If a statement is bigger than this, the thread opens a temporary file to store the transaction. The temporary file is deleted when the thread ends.

Modifications to non-transactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to non-transactional tables, the entire transaction is logged with a [ROLLBACK](#) statement at the end to ensure that the modifications to those

tables are replicated.

The `Binlog_cache_use` status variable shows the number of transactions that used this buffer (and possibly a temporary file) for storing statements. The `Binlog_cache_disk_use` status variable shows how many of those transactions actually had to use a temporary file. These two variables can be used for tuning `binlog_cache_size` to a large enough value that avoids the use of temporary files.

The `max_binlog_cache_size` system variable (default 4GB, which is also the maximum) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this many bytes, it fails and rolls back. The minimum value is 4096.

If you are using the binary log and row based logging, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statement. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. If you are using statement based logging then the original statement is written to the log.

Note that the binary log format is different in MySQL 5.1 from previous versions of MySQL, due to enhancements in replication. See [Section 15.3.2, “Replication Compatibility Between MySQL Versions”](#).

By default, the binary log is not synchronized to disk at each write. So if the operating system or machine (not only the MySQL server) crashes, there is a chance that the last statements of the binary log are lost. To prevent this, you can make the binary log be synchronized to disk after every *N* writes to the binary log, with the `sync_binlog` system variable. See [Section 5.2.3, “System Variables”](#). 1 is the safest value for `sync_binlog`, but also the slowest. Even with `sync_binlog` set to 1, there is still the chance of an inconsistency between the table content and binary log content in case of a crash. For example, if you are using `InnoDB` tables and the MySQL server processes a `COMMIT` statement, it writes the whole transaction to the binary log and then commits this transaction into `InnoDB`. If the server crashes between those two operations, the transaction is rolled back by `InnoDB` at restart but still exists in the binary log. To resolve this, you should set `--innodb_support_xa` to 1. Although this option is related to the support of XA transactions in `InnoDB`, it also ensures that the binary log and `InnoDB` data files are synchronized.

For this option to provide a greater degree of safety, the MySQL server should also be configured to synchronize the binary log and the `InnoDB` logs to disk at every transaction. The `InnoDB` logs are synchronized by default, and `sync_binlog=1` can be used to synchronize the binary log. The effect of this option is that at restart after a crash, after doing a rollback of transactions, the MySQL server cuts rolled back `InnoDB` transactions from the binary log. This ensures that the binary log reflects the exact data of `InnoDB` tables, and so, that the slave remains in synchrony with the master (not receiving a statement which has been rolled back).

If the MySQL server discovers at crash recovery that the binary log is shorter than it should have been, it lacks at least one successfully committed `InnoDB` transaction. This should not happen if `sync_binlog=1` and the disk/filesystem do an actual sync when they are requested to (some don't), so the server prints an error message `The binary log <name> is shorter than its expected size`. In this case, this binary log is not correct and replication should be restarted from a fresh snapshot of the master's data.

For MySQL 5.1.20 and later (and MySQL 5.0.46 for backwards compatibility), the following session variables are written to the binary log and honoured by the replication slave when parsing the binary log:

- `SQL_MODE`
- `FOREIGN_KEY_CHECKS`
- `UNIQUE_CHECKS`
- `CHARACTER_SET_CLIENT`
- `COLLATION_CONNECTION`
- `COLLATION_DATABASE`
- `COLLATION_SERVER`
- `SQL_AUTO_IS_NULL`

5.11.4.1. Binary Logging Formats

A number of different logging formats are used to record information in the binary log. The exact format employed depends on the version of MySQL being used. There are three logging formats:

- Replication capabilities in MySQL originally were based on propagation of SQL statements from master to slave. This is called

statement-based logging (SBL). SBL is identified internally using [STATEMENT](#).

- In *row-based logging* (RBL), the master writes events to the binary log that indicate how individual table rows are affected. Support for RBL was added in MySQL 5.1.5. RBL is identified internally using [ROW](#).
- As of MySQL 5.1.8, a third option is available: *mixed-based logging* (MBL). With MBL, SBL is used by default, but automatically switches to RBL in particular cases as described below. From MySQL 5.1.8, MBL is the default mode. MBL is identified internally using [MIXED](#).

Starting with MySQL 5.1.20, MBL is the default but storage engine being used can also set or limit the logging format according to the capabilities of the storage engine. This helps to eliminate issues when logging, and more specifically replicating, certain statements between the master and slave using the different engines.

5.11.4.2. Setting The Binary Log Format

The default binary logging format depends on the version of MySQL you are using:

- For MySQL 5.1.11 and earlier, statement-based logging is used by default.
- For MySQL 5.1.12 and later, mixed-based logging is used by default.

You can force the default replication format by specifying the format type to the `--binlog-format=type` option. When set, all replication slaves connecting to the server will read the events according to this setting. The supported options are:

- [ROW](#) — sets row-based replication as the default.
- [STATEMENT](#) — sets statement-based replication as the default. This is the default for MySQL 5.1.11 and earlier.
- [MIXED](#) — sets mixed-based replication as the default. This is the default for MySQL 5.1.12 and later.

The logging format also can be switched at runtime. To specify the format globally for all clients, set the global value of the `binlog_format` system variable. (To change a global variable you need the [SUPER](#) privilege.)

To switch to statement-based format, use either of these statements:

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';
mysql> SET GLOBAL binlog_format = 1;
```

To switch to row-based format, use either of these statements:

```
mysql> SET GLOBAL binlog_format = 'ROW';
mysql> SET GLOBAL binlog_format = 2;
```

To switch to mixed format, use either of these statements:

```
mysql> SET GLOBAL binlog_format = 'MIXED';
mysql> SET GLOBAL binlog_format = 3;
```

Individual clients can control the logging format for their own statements by setting the session value of `binlog_format`. For example:

```
mysql> SET SESSION binlog_format = 'STATEMENT';
mysql> SET SESSION binlog_format = 'ROW';
mysql> SET SESSION binlog_format = 'MIXED';
```

In addition to switching the logging format manually, a slave server may switch the format *automatically*. This happens when the server is running in either [STATEMENT](#) or [MIXED](#) format and encounters a row in the binary log that is written in [ROW](#) logging format. In that case, the slave switches to row-based replication temporarily for that event, and switches back to the previous format afterwards.

There are two reasons why you might want to set replication logging on a per-connection basis:

- A thread that makes many small changes to the database might want to use row-based logging. A thread that performs updates that match many rows in the `WHERE` clause might want to use statement-based logging because it will be more efficient to log a few statements than many rows.
- Some statements require a lot of execution time on the master, but result in just a few rows being modified. It might therefore be beneficial to replicate them using row-based logging.

There are exceptions when you cannot switch the replication format at runtime:

- From within a stored function or a trigger.
- If `NDB` is enabled.
- If the session is currently in row-based replication mode and has open temporary tables.

Trying to switch the format in those cases results in an error.

Switching the replication format at runtime is not recommended when any *temporary tables* exist, because temporary tables are logged only when using statement-based replication, whereas with row-based replication they are not logged. With mixed replication, temporary tables are usually logged; exceptions happen with user-defined functions (UDF) and with the `UUID()` function.

With the binlog format set to `ROW`, many changes are written to the binary log using the row-based format. Some changes, however, still use the statement-based format. Examples include all DDL (data definition language) statements such as `CREATE TABLE`, `ALTER TABLE`, or `DROP TABLE`.

The `--binlog-row-event-max-size` option is available for servers that are capable of row-based replication. Rows are stored into the binary log in chunks having a size in bytes not exceeding the value of this option. The value must be a multiple of 256. The default value is 1024.

Warning

When using *statement-based logging* in a replication scenario, it is possible for the data on the master and slave to become different if a statement is designed in such a way that the data modification is *non-deterministic*; that is, it is left to the will of the query optimizer. In general, this is not a good practice even outside of replication. For a detailed explanation of this issue, see [Section B.1.8.1, “Open Issues in MySQL”](#).

5.11.4.3. Mixed Binary Logging (MBL) Format

When running in `MIXED` mode, automatic switching from statement-based to row-based replication takes place under the following conditions:

- When a DML statement updates an `NDB` table
- When a function contains `UUID()`
- When 2 or more tables with `AUTO_INCREMENT` columns are updated
- When any `INSERT DELAYED` is executed
- When the body of a view requires row-based replication, the statement creating the view also uses it — for example, this occurs when the statement creating a view uses the `UUID()` function
- When a call to a UDF is involved.

Note

Starting with MySQL 5.1.20 a warning is generated if you try to log execute a statement in statement-logging mode that should be logged in row-logging mode. The warning will be shown both in the client (and available with `SHOW WARNINGS`) and through the `mysqld` error log. A warning is added to the `SHOW WARNINGS` table each time a statement is executed. However, only the first statement that generated the warning for each client session is logged to the `mysqld` error log to prevent flooding the error log.

Starting with MySQL 5.1.20, mixed based logging is used by default, but in addition to the decisions above, individual engines can also enforce the logging format used when information in a table is updated.

The logging formats supported by a particular engine are controlled by internal flags. The table below lists the logging formats supported by each of the current engines.

Engine	Row-logging supported	Statement-logging supported
Archive	Yes	Yes
Blackhole	No	Yes
CSV	Yes	Yes
Example	Yes	No
Federated	Yes	Yes
Heap	Yes	Yes
MyISAM	Yes	Yes
Merge	Yes	Yes
NDB	Yes	No

Note that an engine can be support either or both logging formats, and the logging capabilities of an individual engine can be further defined as follows

- If an engine supports row-based logging, then the engine is said to be *row-logging capable (RLC)*.
- If an engine support statement-based logging, then the engine is said to be *statement-logging capable (SLC)*.

When determining what logging mode to use, the capabilities of all the tables in the event are combined. The set of tables is then marked according to these rules:

- A set of tables is defined as *row logging restricted (RLR)* if the tables are row logging capable but not statement logging capable.
- A set of tables is defined as *statement logging restricted (SLR)* if the tables are statement logging capable but not row logging capable.

Once the determination of the possible logging formats required by the statement is complete it is compared to the current **BIN-LOG-FORMAT** setting. The following table is used to decide how the information is recorded in the binary log or, if appropriate, whether an error is raised. In the table, a safe operation is defined as one that is deterministic. A number of internal rules decide whether the statement is deterministic or not.

Condition				Action	
Safe/unsafe	BIN-LOG-FORMAT	RLC	SLC	Error/Warning	Logged as
Safe	STATE-MENT	N	N	Error: not loggable	
Safe	STATE-MENT	N	Y		STATE-MENT
Safe	STATE-MENT	Y	N	Error: not loggable	
Safe	STATE-MENT	Y	Y		STATE-MENT
Safe	MIXED	N	N	Error: not loggable	
Safe	MIXED	N	Y		STATE-MENT
Safe	MIXED	Y	N		ROW

Condition				Action	
Safe/unsafe	BIN-LOG_FORMAT	RLC	SLC	Error/Warning	Logged as
Safe	MIXED	Y	Y		STATEMENT
Safe	ROW	N	N	Error: not loggable	
Safe	ROW	N	Y	Error: not loggable	
Safe	ROW	Y	N		ROW
Safe	ROW	Y	Y		ROW
Unsafe	STATEMENT	N	N	Error: not loggable	
Unsafe	STATEMENT	N	Y	Warning: unsafe	STATEMENT
Unsafe	STATEMENT	Y	N	Error: not loggable	
Unsafe	STATEMENT	Y	Y	Warning: unsafe	STATEMENT
Unsafe	MIXED	N	N	Error: not loggable	
Unsafe	MIXED	N	Y	Error: not loggable	
Unsafe	MIXED	Y	N		ROW
Unsafe	MIXED	Y	Y		ROW
Unsafe	ROW	N	N	Error: not loggable	
Unsafe	ROW	N	Y	Error: not loggable	
Unsafe	ROW	Y	N		ROW
Unsafe	ROW	Y	Y		ROW

When a warning is produced by the determination, a standard MySQL warning is produced (and is available using [SHOW WARNINGS](#)). The information is also written to the [mysql](#) error log. Only one error for each error instance per client connection is logged. The log message will include the SQL statement that was attempted.

5.11.4.4. Logging Format for Changes to [mysql](#) Database Tables

The contents of the grant tables in the [mysql](#) database can be modified directly (for example, with [INSERT](#) or [DELETE](#)) or indirectly (for example, with [GRANT](#) or [CREATE USER](#)). As of MySQL 5.1.17, statements that affect [mysql](#) database tables are written to the binary log using the following rules:

- Data manipulation statements that change data in [mysql](#) database tables directly are logged according to the setting of the [binlog_format](#) system variable. This pertains to statements such as [INSERT](#), [UPDATE](#), [DELETE](#), [REPLACE](#), [DO](#), [LOAD DATA INFILE](#), [SELECT](#), and [TRUNCATE](#).
- Statements that change the [mysql](#) database indirectly are logged as statements regardless of the value of [binlog_format](#). This pertains to statements such as [GRANT](#), [REVOKE](#), [SET PASSWORD](#), [RENAME USER](#), [CREATE](#) (all forms except [CREATE TABLE ... SELECT](#)), [ALTER](#) (all forms), and [DROP](#) (all forms).

[CREATE TABLE ... SELECT](#) is a combination of data definition and data manipulation. The [CREATE TABLE](#) part is logged using statement format and the [SELECT](#) part is logged according to the value of [binlog_format](#).

5.11.5. The Slow Query Log

The slow query log consists of all SQL statements that took more than [long_query_time](#) seconds to execute. The time to acquire the initial table locks is not counted as execution time. [mysql](#) writes a statement to the slow query log after it has been executed and after all locks have been released, so log order might be different from execution order. The minimum and default values of

`long_query_time` are 1 and 10, respectively.

To enable the slow query log as of MySQL 5.1.6, start `mysqld` with the `--log-slow-queries[=file_name]` option, and optionally use `--log-output` to specify the log destination (as described in [Section 5.11.1, “Selecting General Query and Slow Query Log Output Destinations”](#)). Before 5.1.6, the slow query log destination is always a file. To enable the slow query log file, use the `--log-slow-queries[=file_name]` option.

If no `file_name` value is given for `--log-slow-queries`, the default name is `host_name-slow.log`. If a filename is given, but not as an absolute pathname, the server writes the file in the data directory.

When `--log-slow-queries` is specified, the `--slow-query-log` option also may be given as of MySQL 5.1.12 to specify the initial slow query log state. With no argument or an argument of 0, the option disables the log. If omitted or given with an argument of 1, the option enables the log. If `--log--slow-queries` is not given, `--slow-query-log` has no effect.

The global `slow_query_log` and `slow_query_log_file` system variables provide runtime control over the slow query log. Set `slow_query_log` to 0 (or `OFF`) to disable the log or to 1 (or `ON`) to enable it. Set `general_log_file` to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

When the slow query log is enabled, output is written to any destinations specified by the `--log-output` option or `log_output` system variable. Note that if the destination is `NONE`, no output is written even if the slow query log is enabled. Setting the log filename has no effect on logging if the log destination value does not contain `FILE`.

The slow query log can be used to find queries that take a long time to execute and are therefore candidates for optimization. However, examining a long slow query log can become a difficult task. To make this easier, you can process the slow query log using the `mysqldumpslow` command to summarize the queries that appear in the log. Use `mysqldumpslow --help` to see the options that this command supports.

In MySQL 5.1, queries that do not use indexes are logged in the slow query log if the `--log-queries-not-using-indexes` option is specified. See [Section 5.2.2, “Command Options”](#).

MySQL Enterprise

Excessive table scans are indicative of missing or poorly optimized indexes. Using an advisor specifically designed for the task, the MySQL Enterprise Monitor can identify such problems and offer advice on resolution. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

In MySQL 5.1, the `--log-slow-admin-statements` server option enables you to request logging of slow administrative statements such as `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `ALTER TABLE` to the slow query log.

Queries handled by the query cache are not added to the slow query log, nor are queries that would not benefit from the presence of an index because the table has zero rows or one row.

5.11.6. Server Log Maintenance

MySQL Server can create a number of different log files that make it easy to see what is going on. See [Section 5.11, “MySQL Server Logs”](#). However, you must clean up these files regularly to ensure that the logs do not take up too much disk space.

When using MySQL with logging enabled, you may want to back up and remove old log files from time to time and tell MySQL to start logging to new files. See [Section 5.9.1, “Database Backups”](#).

On a Linux (Red Hat) installation, you can use the `mysql-log-rotate` script for this. If you installed MySQL from an RPM distribution, this script should have been installed automatically. You should be careful with this script if you are using the binary log for replication. You should not remove binary logs until you are certain that their contents have been processed by all slaves.

On other systems, you must install a short script yourself that you start from `cron` (or its equivalent) for handling log files.

For the binary log, you can set the `expire_logs_days` system variable to expire binary log files automatically after a given number of days (see [Section 5.2.3, “System Variables”](#)). If you are using replication, you should set the variable no lower than the maximum number of days your slaves might lag behind the master.

You can force MySQL to start using new log files by issuing a `FLUSH LOGS` statement or executing `mysqladmin flush-logs` or `mysqladmin refresh`. See [Section 12.5.5.2, “FLUSH Syntax”](#), and [Section 7.10, “mysqladmin — Client for Administering a MySQL Server”](#).

A log flushing operation does the following:

- If general query logging (`--log`) or slow query logging (`--log-slow-queries`) to a log file is enabled, the server closes and reopens the general query log file or slow query log file.
- If binary logging (`--log-bin`) is used, the server closes the current log file and opens a new log file with the next sequence number.
- If the server was given an error log filename with the `--log-error` option, it renames the error log with the suffix `-old` and creates a new empty error log file.

The server creates a new binary log file when you flush the logs. However, it just closes and reopens the general and slow query log files. To cause new files to be created on Unix, rename the current logs before flushing them. At flush time, the server will open new logs with the original names. For example, if the general and slow query logs are named `mysql.log` and `mysql-slow.log`, you can use a series of commands like this:

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mv mysql-slow.log mysql-slow.old
shell> mysqladmin flush-logs
```

At this point, you can make a backup of `mysql.old` and `mysql-slow.log` and then remove them from disk.

On Windows, you cannot rename log files while the server has them open. You must stop the server and rename them, and then restart the server to create new logs.

As of MySQL 5.1.2, you can disable the general query log or slow query log at runtime:

```
SET GLOBAL general_log = 'OFF';
SET GLOBAL slow_query_log = 'OFF';
```

With the logs disabled, rename the log files externally; for example, from the command line. Then enable the logs again:

```
SET GLOBAL general_log = 'ON';
SET GLOBAL slow_query_log = 'ON';
```

This method works on any platform and does not require a server restart.

5.12. Running Multiple MySQL Servers on the Same Machine

In some cases, you might want to run multiple `mysqld` servers on the same machine. You might want to test a new MySQL release while leaving your existing production setup undisturbed. Or you might want to give different users access to different `mysqld` servers that they manage themselves. (For example, you might be an Internet Service Provider that wants to provide independent MySQL installations for different customers.)

To run multiple servers on a single machine, each server must have unique values for several operating parameters. These can be set on the command line or in option files. See [Section 4.3, “Specifying Program Options”](#).

At least the following options must be different for each server:

- `--port=port_num`
`--port` controls the port number for TCP/IP connections. (Alternatively, if the host has multiple network addresses, you can use `--bind-address` to cause different servers to listen to different interfaces.)
- `--socket=path`
`--socket` controls the Unix socket file path on Unix and the name of the named pipe on Windows. On Windows, it is necessary to specify distinct pipe names only for those servers that support named-pipe connections.
- `--shared-memory-base-name=name`

This option currently is used only on Windows. It designates the shared-memory name used by a Windows server to allow clients to connect via shared memory. It is necessary to specify distinct shared-memory names only for those servers that support shared-memory connections.

- `--pid-file=file_name`

This option is used only on Unix. It indicates the pathname of the file in which the server writes its process ID.

If you use the following log file options, they must be different for each server:

- `--log=file_name`
- `--log-bin=file_name`
- `--log-update=file_name`
- `--log-error=file_name`

Section 5.11.6, “Server Log Maintenance”, discusses the log file options further.

For better performance, you can specify the following options differently for each server, to spread the load between several physical disks:

- `--tmpdir=path`

Having different temporary directories is also recommended to make it easier to determine which MySQL server created any given temporary file.

With very limited exceptions, each server should use a different data directory, which is specified using the `--datadir=path` option.

Warning

Normally, you should never have two servers that update data in the same databases. This may lead to unpleasant surprises if your operating system does not support fault-free system locking. If (despite this warning) you run multiple servers using the same data directory and they have logging enabled, you must use the appropriate options to specify log filenames that are unique to each server. Otherwise, the servers try to log to the same files. Please note that this kind of setup only works with **MyISAM** and **MERGE** tables, and not with any of the other storage engines.

The warning against sharing a data directory among servers also applies in an NFS environment. Allowing multiple MySQL servers to access a common data directory over NFS is a *very bad idea*.

- The primary problem is that NFS is the speed bottleneck. It is not meant for such use.
- Another risk with NFS is that you must devise a way to ensure that two or more servers do not interfere with each other. Usually NFS file locking is handled by the **lockd** daemon, but at the moment there is no platform that performs locking 100% reliably in every situation.

Make it easy for yourself: Forget about sharing a data directory among servers over NFS. A better solution is to have one computer that contains several CPUs and use an operating system that handles threads efficiently.

If you have multiple MySQL installations in different locations, you can specify the base installation directory for each server with the `--basedir=path` option to cause each server to use a different data directory, log files, and PID file. (The defaults for all these values are determined relative to the base directory). In that case, the only other options you need to specify are the `--socket` and `--port` options. For example, suppose that you install different versions of MySQL using **tar** file binary distributions. These install in different locations, so you can start the server for each installation using the command `bin/mysqld_safe` under its corresponding base directory. `mysqld_safe` determines the proper `--basedir` option to pass to `mysqld`, and you need specify only the `--socket` and `--port` options to `mysqld_safe`.

As discussed in the following sections, it is possible to start additional servers by setting environment variables or by specifying appropriate command-line options. However, if you need to run multiple servers on a more permanent basis, it is more convenient to use option files to specify for each server those option values that must be unique to it. The `--defaults-file` option is useful for this purpose.

5.12.1. Running Multiple Servers on Windows

You can run multiple servers on Windows by starting them manually from the command line, each with appropriate operating parameters. On Windows NT-based systems, you also have the option of installing several servers as Windows services and running them that way. General instructions for running MySQL servers from the command line or as services are given in [Section 2.3, “Installing MySQL on Windows”](#). This section describes how to make sure that you start each server with different values for those startup options that must be unique per server, such as the data directory. These options are described in [Section 5.12, “Running Multiple MySQL Servers on the Same Machine”](#).

5.12.1.1. Starting Multiple Windows Servers at the Command Line

To start multiple servers manually from the command line, you can specify the appropriate options on the command line or in an option file. It is more convenient to place the options in an option file, but it is necessary to make sure that each server gets its own set of options. To do this, create an option file for each server and tell the server the filename with a `--defaults-file` option when you run it.

Suppose that you want to run `mysqld` on port 3307 with a data directory of `C:\mydata1`, and `mysqld-debug` on port 3308 with a data directory of `C:\mydata2`. (To do this, make sure that before you start the servers, each data directory exists and has its own copy of the `mysql` database that contains the grant tables.) Then create two option files. For example, create one file named `C:\my-opts1.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

Create a second file named `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata2
port = 3308
```

Then start each server with its own option file:

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld-debug --defaults-file=C:\my-opts2.cnf
```

On NT, each server starts in the foreground (no new prompt appears until the server exits later), so you will need to issue those two commands in separate console windows.

To shut down the servers, you must connect to each using the appropriate port number:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 shutdown
```

Servers configured as just described allow clients to connect over TCP/IP. If your version of Windows supports named pipes and you also want to allow named-pipe connections, use the `mysqld-nt` or `mysqld-debug` server and specify options that enable the named pipe and specify its name. Each server that supports named-pipe connections must use a unique pipe name. For example, the `C:\my-opts1.cnf` file might be written like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

Then start the server this way:

```
C:\> C:\mysql\bin\mysqld-nt --defaults-file=C:\my-opts1.cnf
```

Modify `C:\my-opts2.cnf` similarly for use by the second server.

A similar procedure applies for servers that you want to support shared-memory connections. Enable such connections with the `-shared-memory` option and specify a unique shared-memory name for each server with the `--shared-memory-base-name` option.

5.12.1.2. Starting Multiple Windows Servers as Services

On NT-based systems, a MySQL server can run as a Windows service. The procedures for installing, controlling, and removing a single MySQL service are described in [Section 2.3.11, “Starting MySQL as a Windows Service”](#).

You can also install multiple MySQL servers as services. In this case, you must make sure that each server uses a different service name in addition to all the other parameters that must be unique for each server.

For the following instructions, assume that you want to run the `mysqld-nt` server from two different versions of MySQL that are installed at `C:\mysql-5.0.19` and `C:\mysql-5.1.21-beta`, respectively. (This might be the case if you're running 5.0.19 as your production server, but also want to conduct tests using 5.1.21-beta.)

The following principles apply when installing a MySQL service with the `--install` or `--install-manual` option:

- If you specify no service name, the server uses the default service name of `MySQL` and the server reads options from the `[mysqld]` group in the standard option files.
- If you specify a service name after the `--install` option, the server ignores the `[mysqld]` option group and instead reads options from the group that has the same name as the service. The server reads options from the standard option files.
- If you specify a `--defaults-file` option after the service name, the server ignores the standard option files and reads options only from the `[mysqld]` group of the named file.

Note

Before MySQL 4.0.17, only a server installed using the default service name (`MySQL`) or one installed explicitly with a service name of `mysqld` will read the `[mysqld]` group in the standard option files. As of 4.0.17, all servers read the `[mysqld]` group if they read the standard option files, even if they are installed using another service name. This allows you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group named after each service for use by the server installed with that service name.

Based on the preceding information, you have several ways to set up multiple services. The following instructions describe some examples. Before trying any of them, be sure that you shut down and remove any existing MySQL services first.

- **Approach 1:** Specify the options for all services in one of the standard option files. To do this, use a different service name for each server. Suppose that you want to run the 5.0.19 `mysqld-nt` using the service name of `mysqld1` and the 5.1.21-beta `mysqld-nt` using the service name `mysqld2`. In this case, you can use the `[mysqld1]` group for 5.0.19 and the `[mysqld2]` group for 5.1.21-beta. For example, you can set up `C:\my.cnf` like this:

```
# options for mysqld1 service
[mysqld1]
basedir = C:/mysql-5.0.19
port = 3307
enable-named-pipe
socket = mypipe1

# options for mysqld2 service
[mysqld2]
basedir = C:/mysql-5.1.21-beta
port = 3308
enable-named-pipe
socket = mypipe2
```

Install the services as follows, using the full server pathnames to ensure that Windows registers the correct executable program for each service:

```
C:\> C:\mysql-5.0.19\bin\mysqld-nt --install mysqld1
C:\> C:\mysql-5.1.21-beta\bin\mysqld-nt --install mysqld2
```

To start the services, use the services manager, or use `NET START` with the appropriate service names:

```
C:\> NET START mysqld1
C:\> NET START mysqld2
```

To stop the services, use the services manager, or use `NET STOP` with the appropriate service names:

```
C:\> NET STOP mysqld1
C:\> NET STOP mysqld2
```

- **Approach 2:** Specify options for each server in separate files and use `--defaults-file` when you install the services to tell each server what file to use. In this case, each file should list options using a `[mysqld]` group.

With this approach, to specify options for the 5.0.19 `mysqld-nt`, create a file `C:\my-opts1.cnf` that looks like this:

```
[mysqld]
basedir = C:/mysql-5.0.19
port = 3307
enable-named-pipe
socket = mypipe1
```

For the 5.1.21-beta `mysqld-nt`, create a file `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
basedir = C:/mysql-5.1.21-beta
port = 3308
enable-named-pipe
socket = mypipe2
```

Install the services as follows (enter each command on a single line):

```
C:\> C:\mysql-5.0.19\bin\mysqld-nt --install mysqld1
--defaults-file=C:\my-opts1.cnf
C:\> C:\mysql-5.1.21-beta\bin\mysqld-nt --install mysqld2
--defaults-file=C:\my-opts2.cnf
```

To use a `--defaults-file` option when you install a MySQL server as a service, you must precede the option with the service name.

After installing the services, start and stop them the same way as in the preceding example.

To remove multiple services, use `mysqld --remove` for each one, specifying a service name following the `--remove` option. If the service name is the default (`MySQL`), you can omit it.

5.12.2. Running Multiple Servers on Unix

The easiest way to run multiple servers on Unix is to compile them with different TCP/IP ports and Unix socket files so that each one is listening on different network interfaces. Compiling in different base directories for each installation also results automatically in a separate, compiled-in data directory, log file, and PID file location for each server.

Assume that an existing 5.0.19 server is configured for the default TCP/IP port number (3306) and Unix socket file (`/tmp/mysql.sock`). To configure a new 5.1.21-beta server to have different operating parameters, use a `configure` command something like this:

```
shell> ./configure --with-tcp-port=port_number \
--with-unix-socket-path=file_name \
--prefix=/usr/local/mysql-5.1.21-beta
```

Here, `port_number` and `file_name` must be different from the default TCP/IP port number and Unix socket file pathname, and the `--prefix` value should specify an installation directory different from the one under which the existing MySQL installation is located.

If you have a MySQL server listening on a given port number, you can use the following command to find out what operating parameters it is using for several important configurable variables, including the base directory and Unix socket filename:

```
shell> mysqladmin --host=host_name --port=port_number variables
```

With the information displayed by that command, you can tell what option values *not* to use when configuring an additional server.

Note that if you specify `localhost` as a hostname, `mysqladmin` defaults to using a Unix socket file connection rather than TCP/IP. From MySQL 4.1 onward, you can explicitly specify the connection protocol to use by using the `-`

`-protocol={TCP|SOCKET|PIPE|MEMORY}` option.

You don't have to compile a new MySQL server just to start with a different Unix socket file and TCP/IP port number. It is also possible to use the same server binary and start each invocation of it with different parameter values at runtime. One way to do so is by using command-line options:

```
shell> mysqld_safe --socket=file_name --port=port_number
```

To start a second server, provide different `--socket` and `--port` option values, and pass a `--datadir=path` option to `mysqld_safe` so that the server uses a different data directory.

Another way to achieve a similar effect is to use environment variables to set the Unix socket filename and TCP/IP port number:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> mysql_install_db --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

This is a quick way of starting a second server to use for testing. The nice thing about this method is that the environment variable settings apply to any client programs that you invoke from the same shell. Thus, connections for those clients are automatically directed to the second server.

Section 2.14, “Environment Variables”, includes a list of other environment variables you can use to affect `mysqld`.

For automatic server execution, the startup script that is executed at boot time should execute the following command once for each server with an appropriate option file path for each command:

```
shell> mysqld_safe --defaults-file=file_name
```

Each option file should contain option values specific to a given server.

On Unix, the `mysqld_multi` script is another way to start multiple servers. See Section 5.3.3, “`mysqld_multi` — Manage Multiple MySQL Servers”.

5.12.3. Using Client Programs in a Multiple-Server Environment

To connect with a client program to a MySQL server that is listening to different network interfaces from those compiled into your client, you can use one of the following methods:

- Start the client with `--host=host_name --port=port_number` to connect via TCP/IP to a remote server, with `-host=127.0.0.1 --port=port_number` to connect via TCP/IP to a local server, or with `--host=localhost --socket=file_name` to connect to a local server via a Unix socket file or a Windows named pipe.
- As of MySQL 4.1, start the client with `--protocol=tcp` to connect via TCP/IP, `--protocol=socket` to connect via a Unix socket file, `--protocol=pipe` to connect via a named pipe, or `--protocol=memory` to connect via shared memory. For TCP/IP connections, you may also need to specify `--host` and `--port` options. For the other types of connections, you may need to specify a `--socket` option to specify a Unix socket file or Windows named-pipe name, or a `--shared-memory-base-name` option to specify the shared-memory name. Shared-memory connections are supported only on Windows.
- On Unix, set the `MYSQL_UNIX_PORT` and `MYSQL_TCP_PORT` environment variables to point to the Unix socket file and TCP/IP port number before you start your clients. If you normally use a specific socket file or port number, you can place commands to set these environment variables in your `.login` file so that they apply each time you log in. See Section 2.14, “Environment Variables”.
- Specify the default Unix socket file and TCP/IP port number in the `[client]` group of an option file. For example, you can use `C:\my.cnf` on Windows, or the `.my.cnf` file in your home directory on Unix. See Section 4.3.2, “Using Option Files”.
- In a C program, you can specify the socket file or port number arguments in the `mysql_real_connect()` call. You can also have the program read option files by calling `mysql_options()`. See Section 25.2.3, “C API Function Descriptions”.
- If you are using the Perl `DBD:mysql` module, you can read options from MySQL option files. For example:

```
$dsn = "DBI:mysql:test;mysql_read_default_group=client;"
. "mysql_read_default_file=/usr/local/mysql/data/my.cnf";
```

```
$dbh = DBI->connect($dsn, $user, $password);
```

See [Section 25.4, “MySQL Perl API”](#).

Other programming interfaces may provide similar capabilities for reading option files.

Chapter 6. Optimization

Optimization is a complex task because ultimately it requires understanding of the entire system to be optimized. Although it may be possible to perform some local optimizations with little knowledge of your system or application, the more optimal you want your system to become, the more you must know about it.

This chapter tries to explain and give some examples of different ways to optimize MySQL. Remember, however, that there are always additional ways to make the system even faster, although they may require increasing effort to achieve.

6.1. Optimization Overview

The most important factor in making a system fast is its basic design. You must also know what kinds of processing your system is doing, and what its bottlenecks are. In most cases, system bottlenecks arise from these sources:

- Disk seeks. It takes time for the disk to find a piece of data. With modern disks, the mean time for this is usually lower than 10ms, so we can in theory do about 100 seeks a second. This time improves slowly with new disks and is very hard to optimize for a single table. The way to optimize seek time is to distribute the data onto more than one disk.
- Disk reading and writing. When the disk is at the correct position, we need to read the data. With modern disks, one disk delivers at least 10–20MB/s throughput. This is easier to optimize than seeks because you can read in parallel from multiple disks.
- CPU cycles. When we have the data in main memory, we need to process it to get our result. Having small tables compared to the amount of memory is the most common limiting factor. But with small tables, speed is usually not the problem.
- Memory bandwidth. When the CPU needs more data than can fit in the CPU cache, main memory bandwidth becomes a bottleneck. This is an uncommon bottleneck for most systems, but one to be aware of.

MySQL Enterprise

For instant notification of system bottlenecks subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

6.1.1. MySQL Design Limitations and Tradeoffs

When using the **MyISAM** storage engine, MySQL uses extremely fast table locking that allows multiple readers or a single writer. The biggest problem with this storage engine occurs when you have a steady stream of mixed updates and slow selects on a single table. If this is a problem for certain tables, you can use another storage engine for them. See [Chapter 13, Storage Engines](#).

MySQL can work with both transactional and non-transactional tables. To make it easier to work smoothly with non-transactional tables (which cannot roll back if something goes wrong), MySQL has the following rules. Note that these rules apply *only* when not running in strict SQL mode or if you use the **IGNORE** specifier for **INSERT** or **UPDATE**.

- All columns have default values.
- If you insert an inappropriate or out-of-range value into a column, MySQL sets the column to the “best possible value” instead of reporting an error. For numerical values, this is 0, the smallest possible value or the largest possible value. For strings, this is either the empty string or as much of the string as can be stored in the column.
- All calculated expressions return a value that can be used instead of signaling an error condition. For example, 1/0 returns **NULL**.

To change the preceding behaviors, you can enable stricter data handling by setting the server SQL mode appropriately. For more information about data handling, see [Section 1.9.6, “How MySQL Deals with Constraints”](#), [Section 5.2.6, “SQL Modes”](#), and [Section 12.2.4, “INSERT Syntax”](#).

6.1.2. Designing Applications for Portability

Because all SQL servers implement different parts of standard SQL, it takes work to write portable database applications. It is very easy to achieve portability for very simple selects and inserts, but becomes more difficult the more capabilities you require. If you want an application that is fast with many database systems, it becomes even more difficult.

All database systems have some weak points. That is, they have different design compromises that lead to different behavior.

To make a complex application portable, you need to determine which SQL servers it must work with, and then determine what features those servers support. You can use the MySQL [crash-me](#) program to find functions, types, and limits that you can use with a selection of database servers. [crash-me](#) does not check for every possible feature, but it is still reasonably comprehensive, performing about 450 tests. An example of the type of information [crash-me](#) can provide is that you should not use column names that are longer than 18 characters if you want to be able to use Informix or DB2.

The [crash-me](#) program and the MySQL benchmarks are all very database independent. By taking a look at how they are written, you can get a feeling for what you must do to make your own applications database independent. The programs can be found in the [sql-bench](#) directory of MySQL source distributions. They are written in Perl and use the DBI database interface. Use of DBI in itself solves part of the portability problem because it provides database-independent access methods. See [Section 6.1.4](#), “The MySQL Benchmark Suite”.

If you strive for database independence, you need to get a good feeling for each SQL server's bottlenecks. For example, MySQL is very fast in retrieving and updating rows for [MyISAM](#) tables, but has a problem in mixing slow readers and writers on the same table. Oracle, on the other hand, has a big problem when you try to access rows that you have recently updated (until they are flushed to disk). Transactional database systems in general are not very good at generating summary tables from log tables, because in this case row locking is almost useless.

MySQL Enterprise

For expert advice on choosing the database engine suitable to your circumstances subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

To make your application *really* database independent, you should define an easily extendable interface through which you manipulate your data. For example, C++ is available on most systems, so it makes sense to use a C++ class-based interface to the databases.

If you use some feature that is specific to a given database system (such as the [REPLACE](#) statement, which is specific to MySQL), you should implement the same feature for other SQL servers by coding an alternative method. Although the alternative might be slower, it enables the other servers to perform the same tasks.

With MySQL, you can use the `/* ! */` syntax to add MySQL-specific keywords to a statement. The code inside `/* */` is treated as a comment (and ignored) by most other SQL servers. For information about writing comments, see [Section 8.5](#), “Comment Syntax”.

If high performance is more important than exactness, as for some Web applications, it is possible to create an application layer that caches all results to give you even higher performance. By letting old results expire after a while, you can keep the cache reasonably fresh. This provides a method to handle high load spikes, in which case you can dynamically increase the cache size and set the expiration timeout higher until things get back to normal.

In this case, the table creation information should contain information about the initial cache size and how often the table should normally be refreshed.

An attractive alternative to implementing an application cache is to use the MySQL query cache. By enabling the query cache, the server handles the details of determining whether a query result can be reused. This simplifies your application. See [Section 6.5.4](#), “The MySQL Query Cache”.

6.1.3. What We Have Used MySQL For

This section describes an early application for MySQL.

During MySQL initial development, the features of MySQL were made to fit our largest customer, which handled data warehousing for a couple of the largest retailers in Sweden.

From all stores, we got weekly summaries of all bonus card transactions, and were expected to provide useful information for the store owners to help them find how their advertising campaigns were affecting their own customers.

The volume of data was quite huge (about seven million summary transactions per month), and we had data for 4–10 years that we needed to present to the users. We got weekly requests from our customers, who wanted instant access to new reports from this data.

We solved this problem by storing all information per month in compressed “transaction tables.” We had a set of simple macros that generated summary tables grouped by different criteria (product group, customer id, store, and so on) from the tables in which the transactions were stored. The reports were Web pages that were dynamically generated by a small Perl script. This script parsed a Web page, executed the SQL statements in it, and inserted the results. We would have used PHP or [mod_perl](#) instead, but they were not available at the time.

For graphical data, we wrote a simple tool in C that could process SQL query results and produce GIF images based on those results. This tool also was dynamically executed from the Perl script that parses the Web pages.

In most cases, a new report could be created simply by copying an existing script and modifying the SQL query that it used. In some cases, we needed to add more columns to an existing summary table or generate a new one. This also was quite simple because we kept all transaction-storage tables on disk. (This amounted to about 50GB of transaction tables and 200GB of other customer data.)

We also let our customers access the summary tables directly with ODBC so that the advanced users could experiment with the data themselves.

This system worked well and we had no problems handling the data with quite modest Sun Ultra SPARCstation hardware (2×200MHz). Eventually the system was migrated to Linux.

6.1.4. The MySQL Benchmark Suite

This benchmark suite is meant to tell any user what operations a given SQL implementation performs well or poorly. You can get a good idea for how the benchmarks work by looking at the code and results in the `sql-bench` directory in any MySQL source distribution.

Note that this benchmark is single-threaded, so it measures the minimum time for the operations performed. We plan to add multi-threaded tests to the benchmark suite in the future.

To use the benchmark suite, the following requirements must be satisfied:

- The benchmark suite is provided with MySQL source distributions. You can either download a released distribution from <http://dev.mysql.com/downloads/>, or use the current development source tree. (See [Section 2.9.3, “Installing from the Development Source Tree”](#).)
- The benchmark scripts are written in Perl and use the Perl DBI module to access database servers, so DBI must be installed. You also need the server-specific DBD drivers for each of the servers you want to test. For example, to test MySQL, PostgreSQL, and DB2, you must have the `DBD:mysql`, `DBD:Pg`, and `DBD:DB2` modules installed. See [Section 2.15, “Perl Installation Notes”](#).

After you obtain a MySQL source distribution, you can find the benchmark suite located in its `sql-bench` directory. To run the benchmark tests, build MySQL, and then change location into the `sql-bench` directory and execute the `run-all-tests` script:

```
shell> cd sql-bench
shell> perl run-all-tests --server=server_name
```

`server_name` should be the name of one of the supported servers. To get a list of all options and supported servers, invoke this command:

```
shell> perl run-all-tests --help
```

The `crash-me` script also is located in the `sql-bench` directory. `crash-me` tries to determine what features a database system supports and what its capabilities and limitations are by actually running queries. For example, it determines:

- What data types are supported
- How many indexes are supported
- What functions are supported
- How big a query can be
- How big a `VARCHAR` column can be

You can find the results from `crash-me` for many different database servers at <http://dev.mysql.com/tech-resources/crash-me.php>. For more information about benchmark results, visit <http://dev.mysql.com/tech-resources/benchmarks/>.

6.1.5. Using Your Own Benchmarks

You should definitely benchmark your application and database to find out where the bottlenecks are. After fixing one bottleneck (or by replacing it with a “dummy” module), you can proceed to identify the next bottleneck. Even if the overall performance for your application currently is acceptable, you should at least make a plan for each bottleneck and decide how to solve it if someday you really need the extra performance.

For examples of portable benchmark programs, look at those in the MySQL benchmark suite. See [Section 6.1.4, “The MySQL Benchmark Suite”](#). You can take any program from this suite and modify it for your own needs. By doing this, you can try different solutions to your problem and test which really is fastest for you.

Another free benchmark suite is the Open Source Database Benchmark, available at <http://osdb.sourceforge.net/>.

It is very common for a problem to occur only when the system is very heavily loaded. We have had many customers who contact us when they have a (tested) system in production and have encountered load problems. In most cases, performance problems turn out to be due to issues of basic database design (for example, table scans are not good under high load) or problems with the operating system or libraries. Most of the time, these problems would be much easier to fix if the systems were not already in production.

To avoid problems like this, you should put some effort into benchmarking your whole application under the worst possible load:

- The `mysqlslap` program can be helpful for simulating a high load produced by multiple clients issuing queries simultaneously. See [Section 7.17, “mysqlslap — Load Emulation Client”](#).
- You can also try Super Smack, available at <http://jeremy.zawodny.com/mysql/super-smack/>.

As suggested by the names of these programs, they can bring a system to its knees, so make sure to use them only on your development systems.

6.2. Optimizing **SELECT** and Other Statements

First, one factor affects all statements: The more complex your permissions setup, the more overhead you have. Using simpler permissions when you issue **GRANT** statements enables MySQL to reduce permission-checking overhead when clients execute statements. For example, if you do not grant any table-level or column-level privileges, the server need not ever check the contents of the `tables_priv` and `columns_priv` tables. Similarly, if you place no resource limits on any accounts, the server does not have to perform resource counting. If you have a very high statement-processing load, it may be worth the time to use a simplified grant structure to reduce permission-checking overhead.

If your problem is with a specific MySQL expression or function, you can perform a timing test by invoking the `BENCHMARK()` function using the `mysql` client program. Its syntax is `BENCHMARK(loop_count, expression)`. The return value is always zero, but `mysql` prints a line displaying approximately how long the statement took to execute. For example:

```
mysql> SELECT BENCHMARK(1000000,1+1);
+-----+
| BENCHMARK(1000000,1+1) |
+-----+
| 0 |
+-----+
1 row in set (0.32 sec)
```

This result was obtained on a Pentium II 400MHz system. It shows that MySQL can execute 1,000,000 simple addition expressions in 0.32 seconds on that system.

All MySQL functions should be highly optimized, but there may be some exceptions. `BENCHMARK()` is an excellent tool for finding out if some function is a problem for your queries.

6.2.1. Optimizing Queries with **EXPLAIN**

```
EXPLAIN tbl_name
```

Or:

```
EXPLAIN [EXTENDED | PARTITIONS] SELECT select_options
```

The **EXPLAIN** statement can be used either as a synonym for **DESCRIBE** or as a way to obtain information about how MySQL executes a **SELECT** statement:

- `EXPLAIN tbl_name` is synonymous with `DESCRIBE tbl_name` or `SHOW COLUMNS FROM tbl_name`.
- When you precede a `SELECT` statement with the keyword `EXPLAIN`, MySQL displays information from the optimizer about the query execution plan. That is, MySQL explains how it would process the `SELECT`, including information about how tables are joined and in which order.
- `EXPLAIN PARTITIONS` is available beginning with MySQL 5.1.5. It is useful only when examining queries involving partitioned tables. For details, see [Section 17.3.4, “Obtaining Information About Partitions”](#).

This section describes the second use of `EXPLAIN` for obtaining query execution plan information. For a description of the `DESCRIBE` and `SHOW COLUMNS` statements, see [Section 12.3.1, “DESCRIBE Syntax”](#), and [Section 12.5.4.4, “SHOW COLUMNS Syntax”](#).

With the help of `EXPLAIN`, you can see where you should add indexes to tables to get a faster `SELECT` that uses indexes to find rows. You can also use `EXPLAIN` to check whether the optimizer joins the tables in an optimal order. To force the optimizer to use a join order corresponding to the order in which the tables are named in the `SELECT` statement, begin the statement with `SELECT STRAIGHT_JOIN` rather than just `SELECT`.

If you have a problem with indexes not being used when you believe that they should be, you should run `ANALYZE TABLE` to update table statistics such as cardinality of keys, that can affect the choices the optimizer makes. See [Section 12.5.2.1, “ANALYZE TABLE Syntax”](#).

`EXPLAIN` returns a row of information for each table used in the `SELECT` statement. The tables are listed in the output in the order that MySQL would read them while processing the query. MySQL resolves all joins using a *single-sweep multi-join* method. This means that MySQL reads a row from the first table, and then finds a matching row in the second table, the third table, and so on. When all tables are processed, MySQL outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

When the `EXTENDED` keyword is used, `EXPLAIN` produces extra information that can be viewed by issuing a `SHOW WARNINGS` statement following the `EXPLAIN` statement. This information displays how the optimizer qualifies table and column names in the `SELECT` statement, what the `SELECT` looks like after the application of rewriting and optimization rules, and possibly other notes about the optimization process. `EXPLAIN EXTENDED` also displays the `filtered` column as of MySQL 5.1.12.

Note

You cannot use the `EXTENDED` and `PARTITIONS` keywords together in the same `EXPLAIN` statement.

Each output row from `EXPLAIN` provides information about one table, and each row contains the following columns:

- `id`

The `SELECT` identifier. This is the sequential number of the `SELECT` within the query.

- `select_type`

The type of `SELECT`, which can be any of those shown in the following table:

<code>SIMPLE</code>	Simple <code>SELECT</code> (not using <code>UNION</code> or subqueries)
<code>PRIMARY</code>	Outermost <code>SELECT</code>
<code>UNION</code>	Second or later <code>SELECT</code> statement in a <code>UNION</code>
<code>DEPENDENT UNION</code>	Second or later <code>SELECT</code> statement in a <code>UNION</code> , dependent on outer query
<code>UNION RESULT</code>	Result of a <code>UNION</code> .
<code>SUBQUERY</code>	First <code>SELECT</code> in subquery
<code>DEPENDENT SUBQUERY</code>	First <code>SELECT</code> in subquery, dependent on outer query
<code>DERIVED</code>	Derived table <code>SELECT</code> (subquery in <code>FROM</code> clause)
<code>UNCACHEABLE SUBQUERY</code>	A subquery for which the result cannot be cached and must be re-evaluated for each row of the outer query
<code>UNCACHEABLE UNION</code>	The second or later select in a <code>UNION</code> that belongs to an uncachable subquery (see <code>UNCACHEABLE SUBQUERY</code>)

DEPENDENT typically signifies the use of a correlated subquery. See [Section 12.2.8.7, “Correlated Subqueries”](#).

“DEPENDENT SUBQUERY” evaluation differs from **UNCACHEABLE SUBQUERY** evaluation. For “DEPENDENT SUBQUERY”, the subquery is re-evaluated only once for each set of different values of the variables from its outer context. For **UNCACHEABLE SUBQUERY**, the subquery is re-evaluated for each row of the outer context. Cacheability of subqueries is subject to the restrictions detailed in [Section 6.5.4.1, “How the Query Cache Operates”](#). For example, referring to user variables makes a subquery uncacheable.

- **table**

The table to which the row of output refers.

- **type**

The join type. The different join types are listed here, ordered from the best type to the worst:

- **system**

The table has only one row (= system table). This is a special case of the **const** join type.

- **const**

The table has at most one matching row, which is read at the start of the query. Because there is only one row, values from the column in this row can be regarded as constants by the rest of the optimizer. **const** tables are very fast because they are read only once.

const is used when you compare all parts of a **PRIMARY KEY** or **UNIQUE** index to constant values. In the following queries, *tbl_name* can be used as a **const** table:

```
SELECT * FROM tbl_name WHERE primary_key=1;

SELECT * FROM tbl_name
WHERE primary_key_part1=1 AND primary_key_part2=2;
```

- **eq_ref**

One row is read from this table for each combination of rows from the previous tables. Other than the **system** and **const** types, this is the best possible join type. It is used when all parts of an index are used by the join and the index is a **PRIMARY KEY** or **UNIQUE** index.

eq_ref can be used for indexed columns that are compared using the **=** operator. The comparison value can be a constant or an expression that uses columns from tables that are read before this table. In the following examples, MySQL can use an **eq_ref** join to process *ref_table*:

```
SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```

- **ref**

All rows with matching index values are read from this table for each combination of rows from the previous tables. **ref** is used if the join uses only a leftmost prefix of the key or if the key is not a **PRIMARY KEY** or **UNIQUE** index (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this is a good join type.

ref can be used for indexed columns that are compared using the **=** or **<=>** operator. In the following examples, MySQL can use a **ref** join to process *ref_table*:

```
SELECT * FROM ref_table WHERE key_column=expr;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```


- **fulltext**

The join is performed using a **FULLTEXT** index.

- **ref_or_null**

This join type is like **ref**, but with the addition that MySQL does an extra search for rows that contain **NULL** values. This join type optimization is used most often in resolving subqueries. In the following examples, MySQL can use a **ref_or_null** join to process **ref_table**:

```
SELECT * FROM ref_table
WHERE key_column=expr OR key_column IS NULL;
```

See [Section 6.2.7, “IS NULL Optimization”](#).

- **index_merge**

This join type indicates that the Index Merge optimization is used. In this case, the **key** column in the output row contains a list of indexes used, and **key_len** contains a list of the longest key parts for the indexes used. For more information, see [Section 6.2.6, “Index Merge Optimization”](#).

- **unique_subquery**

This type replaces **ref** for some **IN** subqueries of the following form:

```
value IN (SELECT primary_key FROM single_table WHERE some_expr)
```

unique_subquery is just an index lookup function that replaces the subquery completely for better efficiency.

- **index_subquery**

This join type is similar to **unique_subquery**. It replaces **IN** subqueries, but it works for non-unique indexes in subqueries of the following form:

```
value IN (SELECT key_column FROM single_table WHERE some_expr)
```

- **range**

Only rows that are in a given range are retrieved, using an index to select the rows. The **key** column in the output row indicates which index is used. The **key_len** contains the longest key part that was used. The **ref** column is **NULL** for this type.

range can be used when a key column is compared to a constant using any of the **=**, **<>**, **>**, **>=**, **<**, **<=**, **IS NULL**, **<=>**, **BETWEEN**, or **IN** operators:

```
SELECT * FROM tbl_name
WHERE key_column = 10;

SELECT * FROM tbl_name
WHERE key_column BETWEEN 10 and 20;

SELECT * FROM tbl_name
WHERE key_column IN (10,20,30);

SELECT * FROM tbl_name
WHERE key_part1= 10 AND key_part2 IN (10,20,30);
```

- **index**

This join type is the same as **ALL**, except that only the index tree is scanned. This usually is faster than **ALL** because the index file usually is smaller than the data file.

MySQL can use this join type when the query uses only columns that are part of a single index.

- **ALL**

A full table scan is done for each combination of rows from the previous tables. This is normally not good if the table is the first

table not marked `const`, and usually very bad in all other cases. Normally, you can avoid `ALL` by adding indexes that allow row retrieval from the table based on constant values or column values from earlier tables.

- `possible_keys`

The `possible_keys` column indicates which indexes MySQL can choose from use to find the rows in this table. Note that this column is totally independent of the order of the tables as displayed in the output from `EXPLAIN`. That means that some of the keys in `possible_keys` might not be usable in practice with the generated table order.

If this column is `NULL`, there are no relevant indexes. In this case, you may be able to improve the performance of your query by examining the `WHERE` clause to check whether it refers to some column or columns that would be suitable for indexing. If so, create an appropriate index and check the query with `EXPLAIN` again. See [Section 12.1.4, “ALTER TABLE Syntax”](#).

To see what indexes a table has, use `SHOW INDEX FROM tbl_name`.

- `key`

The `key` column indicates the key (index) that MySQL actually decided to use. If MySQL decides to use one of the `possible_keys` indexes to look up rows, that index is listed as the key value.

It is possible that `key` will name an index that is not present in the `possible_keys` value. This can happen if none of the `possible_keys` indexes are suitable for looking up rows, but all the columns selected by the query are columns of some other index. That is, the named index covers the selected columns, so although it is not used to determine which rows to retrieve, an index scan is more efficient than a data row scan.

For `InnoDB`, a secondary index might cover the selected columns even if the query also selects the primary key because `InnoDB` stores the primary key value with each secondary index. If `key` is `NULL`, MySQL found no index to use for executing the query more efficiently.

To force MySQL to use or ignore an index listed in the `possible_keys` column, use `FORCE INDEX`, `USE INDEX`, or `IGNORE INDEX` in your query. See [Section 12.2.7.2, “Index Hint Syntax”](#).

For `MyISAM` tables, running `ANALYZE TABLE` helps the optimizer choose better indexes. For `MyISAM` tables, `myisamchk -analyze` does the same. See [Section 12.5.2.1, “ANALYZE TABLE Syntax”](#), and [Section 5.9.4, “Table Maintenance and Crash Recovery”](#).

- `key_len`

The `key_len` column indicates the length of the key that MySQL decided to use. The length is `NULL` if the `key` column says `NULL`. Note that the value of `key_len` enables you to determine how many parts of a multiple-part key MySQL actually uses.

- `ref`

The `ref` column shows which columns or constants are compared to the index named in the `key` column to select rows from the table.

- `rows`

The `rows` column indicates the number of rows MySQL believes it must examine to execute the query.

- `filtered`

The `filtered` column indicates an estimated percentage of table rows that will be filtered by the table condition. That is, `rows` shows the estimated number of rows examined and `rows × filtered / 100` shows the number of rows that will be joined with previous tables. This column is displayed if you use `EXPLAIN EXTENDED`. (New in MySQL 5.1.12)

- `Extra`

This column contains additional information about how MySQL resolves the query. The following list explains the values that can appear in this column. If you want to make your queries as fast as possible, you should look out for `Extra` values of `Using filesort` and `Using temporary`.

- `Distinct`

MySQL is looking for distinct values, so it stops searching for more rows for the current row combination after it has found the first matching row.

- [Full scan on NULL key](#)

This occurs for subquery optimization as a fallback strategy when the optimizer cannot use an index-lookup access method.

- [Impossible WHERE noticed after reading const tables](#)

MySQL has read all [const](#) (and [system](#)) tables and notice that the [WHERE](#) clause is always false.

- [No tables](#)

The query has no [FROM](#) clause, or has a [FROM DUAL](#) clause.

- [Not exists](#)

MySQL was able to do a [LEFT JOIN](#) optimization on the query and does not examine more rows in this table for the previous row combination after it finds one row that matches the [LEFT JOIN](#) criteria. Here is an example of the type of query that can be optimized this way:

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
WHERE t2.id IS NULL;
```

Assume that [t2.id](#) is defined as [NOT NULL](#). In this case, MySQL scans [t1](#) and looks up the rows in [t2](#) using the values of [t1.id](#). If MySQL finds a matching row in [t2](#), it knows that [t2.id](#) can never be [NULL](#), and does not scan through the rest of the rows in [t2](#) that have the same [id](#) value. In other words, for each row in [t1](#), MySQL needs to do only a single lookup in [t2](#), regardless of how many rows actually match in [t2](#).

- [range checked for each record \(index map: N\)](#)

MySQL found no good index to use, but found that some of indexes might be used after column values from preceding tables are known. For each row combination in the preceding tables, MySQL checks whether it is possible to use a [range](#) or [index_merge](#) access method to retrieve rows. This is not very fast, but is faster than performing a join with no index at all. The applicability criteria are as described in [Section 6.2.5, “Range Optimization”](#), and [Section 6.2.6, “Index Merge Optimization”](#), with the exception that all column values for the preceding table are known and considered to be constants.

- [Select tables optimized away](#)

The query contained only aggregate functions ([MIN\(\)](#), [MAX\(\)](#)) that were all resolved using an index, or [COUNT\(*\)](#) for [MyISAM](#), and no [GROUP BY](#) clause. The optimizer determined that only one row should be returned.

- [Using filesort](#)

MySQL must do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the [WHERE](#) clause. The keys then are sorted and the rows are retrieved in sorted order. See [Section 6.2.11, “ORDER BY Optimization”](#).

- [Using index](#)

The column information is retrieved from the table using only information in the index tree without having to do an additional seek to read the actual row. This strategy can be used when the query uses only columns that are part of a single index.

- [Using join buffer](#)

Tables are read in portions into the join buffer, and then their rows are used from the buffer to perform the join.

- [Using temporary](#)

To resolve the query, MySQL needs to create a temporary table to hold the result. This typically happens if the query contains [GROUP BY](#) and [ORDER BY](#) clauses that list columns differently.

- [Using where](#)

A **WHERE** clause is used to restrict which rows to match against the next table or send to the client. Unless you specifically intend to fetch or examine all rows from the table, you may have something wrong in your query if the **Extra** value is not **Using where** and the table join type is **ALL** or **index**.

- **Using sort_union(...), Using union(...), Using intersect(...)**

These indicate how index scans are merged for the **index_merge** join type. See [Section 6.2.6, “Index Merge Optimization”](#), for more information.

- **Using index for group-by**

Similar to the **Using index** way of accessing a table, **Using index for group-by** indicates that MySQL found an index that can be used to retrieve all columns of a **GROUP BY** or **DISTINCT** query without any extra disk access to the actual table. Additionally, the index is used in the most efficient way so that for each group, only a few index entries are read. For details, see [Section 6.2.12, “GROUP BY Optimization”](#).

- **Using where with pushed condition**

This item applies to **NDB Cluster** tables *only*. It means that MySQL Cluster is using *condition pushdown* to improve the efficiency of a direct comparison (**=**) between a non-indexed column and a constant. In such cases, the condition is “pushed down” to the cluster's data nodes where it is evaluated in all partitions simultaneously. This eliminates the need to send non-matching rows over the network, and can speed up such queries by a factor of 5 to 10 times over cases where condition pushdown could be but is not used.

Suppose that you have a Cluster table defined as follows:

```
CREATE TABLE t1 (
  a INT,
  b INT,
  KEY(a)
) ENGINE=NDBCLUSTER;
```

In this case, condition pushdown can be used with a query such as this one:

```
SELECT a,b FROM t1 WHERE b = 10;
```

This can be seen in the output of **EXPLAIN SELECT**, as shown here:

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE b = 10\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: t1
        type: ALL
possible_keys: NULL
         key: NULL
        key_len: NULL
         ref: NULL
          rows: 10
    Extra: Using where with pushed condition
```

Condition pushdown *cannot* be used with either of these two queries:

```
SELECT a,b FROM t1 WHERE a = 10;
SELECT a,b FROM t1 WHERE b + 1 = 10;
```

With regard to the first of these two queries, condition pushdown is not applicable because an index exists on column **a**. In the case of the second query, a condition pushdown cannot be employed because the comparison involving the non-indexed column **b** is an indirect one. (However, it would apply if you were to reduce **b + 1 = 10** to **b = 9** in the **WHERE** clause.)

However, a condition pushdown may also be employed when an indexed column is compared with a constant using a **>** or **<** operator:

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE a<2\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: t1
```

```

    type: range
possible_keys: a
  key: a
  key_len: 5
  ref: NULL
  rows: 2
  Extra: Using where with pushed condition

```

With regard to condition pushdown, keep in mind that:

- Condition pushdown is relevant to MySQL Cluster *only*, and does not occur when executing queries against tables using any other storage engine.
- Condition pushdown capability is not used by default. To enable it, you can start `mysqld` with the `-engine-condition-pushdown` option, or execute the following statement:

```
SET engine_condition_pushdown=0n;
```

Note

Condition pushdown is not supported for columns of any of the `BLOB` or `TEXT` types.

You can get a good indication of how good a join is by taking the product of the values in the `rows` column of the `EXPLAIN` output. This should tell you roughly how many rows MySQL must examine to execute the query. If you restrict queries with the `max_join_size` system variable, this row product also is used to determine which multiple-table `SELECT` statements to execute and which to abort. See [Section 6.5.2, “Tuning Server Parameters”](#).

The following example shows how a multiple-table join can be optimized progressively based on the information provided by `EXPLAIN`.

Suppose that you have the `SELECT` statement shown here and that you plan to examine it using `EXPLAIN`:

```

EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
               tt.ProjectReference, tt.EstimatedShipDate,
               tt.ActualShipDate, tt.ClientID,
               tt.ServiceCodes, tt.RepetitiveID,
               tt.CurrentProcess, tt.CurrentDPPerson,
               tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
               et_1.COUNTRY, do.CUSTNAME
FROM tt, et, et AS et_1, do
WHERE tt.SubmitTime IS NULL
      AND tt.ActualPC = et.EMPLOYID
      AND tt.AssignedPC = et_1.EMPLOYID
      AND tt.ClientID = do.CUSTNMBR;

```

For this example, make the following assumptions:

- The columns being compared have been declared as follows:

Table	Column	Data Type
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- The tables have the following indexes:

Table	Index
tt	ActualPC
tt	AssignedPC

tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- The `tt.ActualPC` values are not evenly distributed.

Initially, before any optimizations have been performed, the `EXPLAIN` statement produces the following information:

```
table type possible_keys key key_len ref rows Extra
et ALL PRIMARY NULL NULL NULL 74
do ALL PRIMARY NULL NULL NULL 2135
et_1 ALL PRIMARY NULL NULL NULL 74
tt ALL AssignedPC, ClientID, ActualPC NULL NULL NULL 3872
range checked for each record (key map: 35)
```

Because `type` is `ALL` for each table, this output indicates that MySQL is generating a Cartesian product of all the tables; that is, every combination of rows. This takes quite a long time, because the product of the number of rows in each table must be examined. For the case at hand, this product is $74 \times 2135 \times 74 \times 3872 = 45,268,558,720$ rows. If the tables were bigger, you can only imagine how long it would take.

One problem here is that MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. `tt.ActualPC` is declared as `CHAR(10)` and `et.EMPLOYID` is `CHAR(15)`, so there is a length mismatch.

To fix this disparity between column lengths, use `ALTER TABLE` to lengthen `ActualPC` from 10 characters to 15 characters:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Now `tt.ActualPC` and `et.EMPLOYID` are both `VARCHAR(15)`. Executing the `EXPLAIN` statement again produces this result:

```
table type possible_keys key key_len ref rows Extra
tt ALL AssignedPC, ClientID, ActualPC NULL NULL NULL 3872 Using where
do ALL PRIMARY NULL NULL NULL 2135
et_1 ALL PRIMARY NULL NULL NULL 74
et eq_ref PRIMARY PRIMARY 15 tt.ActualPC 1
range checked for each record (key map: 1)
```

This is not perfect, but is much better: The product of the `rows` values is less by a factor of 74. This version executes in a couple of seconds.

A second alteration can be made to eliminate the column length mismatches for the `tt.AssignedPC = et_1.EMPLOYID` and `tt.ClientID = do.CUSTNMBR` comparisons:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
-> MODIFY ClientID VARCHAR(15);
```

After that modification, `EXPLAIN` produces the output shown here:

```
table type possible_keys key key_len ref rows Extra
et ALL PRIMARY NULL NULL NULL 74
tt ref AssignedPC, ClientID, ActualPC NULL 15 et.EMPLOYID 52 Using where
et_1 eq_ref PRIMARY PRIMARY 15 tt.AssignedPC 1
do eq_ref PRIMARY PRIMARY 15 tt.ClientID 1
```

At this point, the query is optimized almost as well as possible. The remaining problem is that, by default, MySQL assumes that values in the `tt.ActualPC` column are evenly distributed, and that is not the case for the `tt` table. Fortunately, it is easy to tell MySQL to analyze the key distribution:

```
mysql> ANALYZE TABLE tt;
```

With the additional index information, the join is perfect and **EXPLAIN** produces this result:

table	type	possible_keys	key	key_len	ref	rows	Extra
tt	ALL	AssignedPC, ClientID, ActualPC	NULL	NULL	NULL	3872	Using where
et	eq_ref	PRIMARY	PRIMARY	15	tt.ActualPC	1	
et_1	eq_ref	PRIMARY	PRIMARY	15	tt.AssignedPC	1	
do	eq_ref	PRIMARY	PRIMARY	15	tt.ClientID	1	

Note that the **rows** column in the output from **EXPLAIN** is an educated guess from the MySQL join optimizer. You should check whether the numbers are even close to the truth by comparing the **rows** product with the actual number of rows that the query returns. If the numbers are quite different, you might get better performance by using **STRAIGHT_JOIN** in your **SELECT** statement and trying to list the tables in a different order in the **FROM** clause.

MySQL Enterprise

Subscribers to the MySQL Enterprise Monitor regularly receive expert advice on optimization. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

6.2.2. Estimating Query Performance

In most cases, you can estimate query performance by counting disk seeks. For small tables, you can usually find a row in one disk seek (because the index is probably cached). For bigger tables, you can estimate that, using B-tree indexes, you need this many seeks to find a row: $\log(\text{row\_count}) / \log(\text{index\_block\_length} / 3 \times 2 / (\text{index\_length} + \text{data\_pointer\_length})) + 1$.

In MySQL, an index block is usually 1,024 bytes and the data pointer is usually four bytes. For a 500,000-row table with an index length of three bytes (the size of **MEDIUMINT**), the formula indicates $\log(500,000) / \log(1024/3 \times 2 / (3+4)) + 1 = 4$ seeks.

This index would require storage of about $500,000 \times 7 \times 3/2 = 5.2\text{MB}$ (assuming a typical index buffer fill ratio of 2/3), so you probably have much of the index in memory and so need only one or two calls to read data to find the row.

For writes, however, you need four seek requests to find where to place a new index value and normally two seeks to update the index and write the row.

Note that the preceding discussion does not mean that your application performance slowly degenerates by $\log N$. As long as everything is cached by the OS or the MySQL server, things become only marginally slower as the table gets bigger. After the data gets too big to be cached, things start to go much slower until your applications are bound only by disk seeks (which increase by $\log N$). To avoid this, increase the key cache size as the data grows. For **MyISAM** tables, the key cache size is controlled by the **key_buffer_size** system variable. See [Section 6.5.2, “Tuning Server Parameters”](#).

MySQL Enterprise

The MySQL Enterprise Monitor provides a number of advisors specifically designed to improve query performance. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

6.2.3. Speed of **SELECT** Queries

In general, when you want to make a slow **SELECT ... WHERE** query faster, the first thing to check is whether you can add an index. All references between different tables should usually be done with indexes. You can use the **EXPLAIN** statement to determine which indexes are used for a **SELECT**. See [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#), and [Section 6.4.5, “How MySQL Uses Indexes”](#).

Some general tips for speeding up queries on **MyISAM** tables:

- To help MySQL better optimize queries, use **ANALYZE TABLE** or run **myisamchk --analyze** on a table after it has been loaded with data. This updates a value for each index part that indicates the average number of rows that have the same value. (For unique indexes, this is always 1.) MySQL uses this to decide which index to choose when you join two tables based on a non-constant expression. You can check the result from the table analysis by using **SHOW INDEX FROM *tbl_name*** and examining the **Cardinality** value. **myisamchk --description --verbose** shows index distribution information.
- To sort an index and data according to an index, use **myisamchk --sort-index --sort-records=1** (assuming that you want to sort on index 1). This is a good way to make queries faster if you have a unique index from which you want to read all rows in order according to the index. The first time you sort a large table this way, it may take a long time.

6.2.4. WHERE Clause Optimization

This section discusses optimizations that can be made for processing **WHERE** clauses. The examples use **SELECT** statements, but the same optimizations apply for **WHERE** clauses in **DELETE** and **UPDATE** statements.

Work on the MySQL optimizer is ongoing, so this section is incomplete. MySQL performs a great many optimizations, not all of which are documented here.

Some of the optimizations performed by MySQL follow:

- Removal of unnecessary parentheses:

```
((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- Constant folding:

```
(a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Constant condition removal (needed because of constant folding):

```
(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- Constant expressions used by indexes are evaluated only once.
- COUNT(*)** on a single table without a **WHERE** is retrieved directly from the table information for **MyISAM** and **MEMORY** tables. This is also done for any **NOT NULL** expression when used with only one table.
- Early detection of invalid constant expressions. MySQL quickly detects that some **SELECT** statements are impossible and returns no rows.
- HAVING** is merged with **WHERE** if you do not use **GROUP BY** or aggregate functions (**COUNT()**, **MIN()**, and so on).
- For each table in a join, a simpler **WHERE** is constructed to get a fast **WHERE** evaluation for the table and also to skip rows as soon as possible.
- All constant tables are read first before any other tables in the query. A constant table is any of the following:
 - An empty table or a table with one row.
 - A table that is used with a **WHERE** clause on a **PRIMARY KEY** or a **UNIQUE** index, where all index parts are compared to constant expressions and are defined as **NOT NULL**.

All of the following tables are used as constant tables:

```
SELECT * FROM t WHERE primary_key=1;
SELECT * FROM t1,t2
WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- The best join combination for joining the tables is found by trying all possibilities. If all columns in **ORDER BY** and **GROUP BY** clauses come from the same table, that table is preferred first when joining.
- If there is an **ORDER BY** clause and a different **GROUP BY** clause, or if the **ORDER BY** or **GROUP BY** contains columns from tables other than the first table in the join queue, a temporary table is created.
- If you use the **SQL_SMALL_RESULT** option, MySQL uses an in-memory temporary table.
- Each table index is queried, and the best index is used unless the optimizer believes that it is more efficient to use a table scan. At one time, a scan was used based on whether the best index spanned more than 30% of the table, but a fixed percentage no longer determines the choice between using an index or a scan. The optimizer now is more complex and bases its estimate on additional factors such as table size, number of rows, and I/O block size.
- In some cases, MySQL can read rows from the index without even consulting the data file. If all columns used from the index are

numeric, only the index tree is used to resolve the query.

- Before each row is output, those that do not match the **HAVING** clause are skipped.

Some examples of queries that are very fast:

```
SELECT COUNT(*) FROM tbl_name;
SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;
SELECT MAX(key_part2) FROM tbl_name
  WHERE key_part1=constant;
SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... LIMIT 10;
SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

MySQL resolves the following queries using only the index tree, assuming that the indexed columns are numeric:

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;
SELECT COUNT(*) FROM tbl_name
  WHERE key_part1=val1 AND key_part2=val2;
SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use indexing to retrieve the rows in sorted order without a separate sorting pass:

```
SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... ;
SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

6.2.5. Range Optimization

The **range** access method uses a single index to retrieve a subset of table rows that are contained within one or several index value intervals. It can be used for a single-part or multiple-part index. The following sections give a detailed description of how intervals are extracted from the **WHERE** clause.

6.2.5.1. The Range Access Method for Single-Part Indexes

For a single-part index, index value intervals can be conveniently represented by corresponding conditions in the **WHERE** clause, so we speak of *range conditions* rather than “intervals.”

The definition of a range condition for a single-part index is as follows:

- For both **BTREE** and **HASH** indexes, comparison of a key part with a constant value is a range condition when using the **=**, **<=>**, **IN**, **IS NULL**, or **IS NOT NULL** operators.
- For **BTREE** indexes, comparison of a key part with a constant value is a range condition when using the **>**, **<**, **>=**, **<=**, **BETWEEN**, **!=**, or **<>** operators, or **LIKE 'pattern'** (where **'pattern'** does not start with a wildcard).
- For all types of indexes, multiple range conditions combined with **OR** or **AND** form a range condition.

“Constant value” in the preceding descriptions means one of the following:

- A constant from the query string
- A column of a **const** or **system** table from the same join
- The result of an uncorrelated subquery

- Any expression composed entirely from subexpressions of the preceding types

Here are some examples of queries with range conditions in the `WHERE` clause:

```
SELECT * FROM t1
WHERE key_col > 1
AND key_col < 10;

SELECT * FROM t1
WHERE key_col = 1
OR key_col IN (15,18,20);

SELECT * FROM t1
WHERE key_col LIKE 'ab%'
OR key_col BETWEEN 'bar' AND 'foo';
```

Note that some non-constant values may be converted to constants during the constant propagation phase.

MySQL tries to extract range conditions from the `WHERE` clause for each of the possible indexes. During the extraction process, conditions that cannot be used for constructing the range condition are dropped, conditions that produce overlapping ranges are combined, and conditions that produce empty ranges are removed.

Consider the following statement, where `key1` is an indexed column and `nonkey` is not indexed:

```
SELECT * FROM t1 WHERE
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z');
```

The extraction process for key `key1` is as follows:

- Start with original `WHERE` clause:

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z')
```

- Remove `nonkey = 4` and `key1 LIKE '%b'` because they cannot be used for a range scan. The correct way to remove them is to replace them with `TRUE`, so that we do not miss any matching rows when doing the range scan. Having replaced them with `TRUE`, we get:

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
(key1 < 'bar' AND TRUE) OR
(key1 < 'uux' AND key1 > 'z')
```

- Collapse conditions that are always true or false:

- `(key1 LIKE 'abcde%' OR TRUE)` is always true
- `(key1 < 'uux' AND key1 > 'z')` is always false

Replacing these conditions with constants, we get:

```
(key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
```

Removing unnecessary `TRUE` and `FALSE` constants, we obtain:

```
(key1 < 'abc') OR (key1 < 'bar')
```

- Combining overlapping intervals into one yields the final condition to be used for the range scan:

```
(key1 < 'bar')
```

In general (and as demonstrated by the preceding example), the condition used for a range scan is less restrictive than the `WHERE` clause. MySQL performs an additional check to filter out rows that satisfy the range condition but not the full `WHERE` clause.

The range condition extraction algorithm can handle nested **AND/OR** constructs of arbitrary depth, and its output does not depend on the order in which conditions appear in **WHERE** clause.

Currently, MySQL does not support merging multiple ranges for the **range** access method for spatial indexes. To work around this limitation, you can use a **UNION** with identical **SELECT** statements, except that you put each spatial predicate in a different **SELECT**.

6.2.5.2. The Range Access Method for Multiple-Part Indexes

Range conditions on a multiple-part index are an extension of range conditions for a single-part index. A range condition on a multiple-part index restricts index rows to lie within one or several key tuple intervals. Key tuple intervals are defined over a set of key tuples, using ordering from the index.

For example, consider a multiple-part index defined as **key1(key_part1, key_part2, key_part3)**, and the following set of key tuples listed in key order:

key_part1	key_part2	key_part3
NULL	1	'abc'
NULL	1	'xyz'
NULL	2	'foo'
1	1	'abc'
1	1	'xyz'
1	2	'abc'
2	1	'aaa'

The condition **key_part1 = 1** defines this interval:

```
(1, -inf, -inf) <= (key_part1, key_part2, key_part3) < (1, +inf, +inf)
```

The interval covers the 4th, 5th, and 6th tuples in the preceding data set and can be used by the range access method.

By contrast, the condition **key_part3 = 'abc'** does not define a single interval and cannot be used by the range access method.

The following descriptions indicate how range conditions work for multiple-part indexes in greater detail.

- For **HASH** indexes, each interval containing identical values can be used. This means that the interval can be produced only for conditions in the following form:

```
key_part1 cmp const1
AND key_part2 cmp const2
AND ...
AND key_partN cmp constN;
```

Here, **const1**, **const2**, ... are constants, **cmp** is one of the **=**, **<=>**, or **IS NULL** comparison operators, and the conditions cover all index parts. (That is, there are **N** conditions, one for each part of an **N**-part index.) For example, the following is a range condition for a three-part **HASH** index:

```
key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
```

For the definition of what is considered to be a constant, see [Section 6.2.5.1, “The Range Access Method for Single-Part Indexes”](#).

- For a **BTREE** index, an interval might be usable for conditions combined with **AND**, where each condition compares a key part with a constant value using **=**, **<=>**, **IS NULL**, **>**, **<**, **>=**, **<=**, **!=**, **<>**, **BETWEEN**, or **LIKE 'pattern'** (where **'pattern'** does not start with a wildcard). An interval can be used as long as it is possible to determine a single key tuple containing all rows that match the condition (or two intervals if **<>** or **!=** is used). For example, for this condition:

```
key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
```

The single interval is:

```
('foo', 10, 10) < (key_part1, key_part2, key_part3) < ('foo', +inf, +inf)
```

It is possible that the created interval contains more rows than the initial condition. For example, the preceding interval includes the value **('foo', 11, 0)**, which does not satisfy the original condition.

- If conditions that cover sets of rows contained within intervals are combined with **OR**, they form a condition that covers a set of

rows contained within the union of their intervals. If the conditions are combined with **AND**, they form a condition that covers a set of rows contained within the intersection of their intervals. For example, for this condition on a two-part index:

```
(key_part1 = 1 AND key_part2 < 2) OR (key_part1 > 5)
```

The intervals are:

```
(1, -inf) < (key_part1, key_part2) < (1, 2)
(5, -inf) < (key_part1, key_part2)
```

In this example, the interval on the first line uses one key part for the left bound and two key parts for the right bound. The interval on the second line uses only one key part. The **key_len** column in the **EXPLAIN** output indicates the maximum length of the key prefix used.

In some cases, **key_len** may indicate that a key part was used, but that might be not what you would expect. Suppose that **key_part1** and **key_part2** can be **NULL**. Then the **key_len** column displays two key part lengths for the following condition:

```
key_part1 >= 1 AND key_part2 < 2
```

But, in fact, the condition is converted to this:

```
key_part1 >= 1 AND key_part2 IS NOT NULL
```

Section 6.2.5.1, “The Range Access Method for Single-Part Indexes”, describes how optimizations are performed to combine or eliminate intervals for range conditions on a single-part index. Analogous steps are performed for range conditions on multiple-part indexes.

6.2.6. Index Merge Optimization

The *Index Merge* method is used to retrieve rows with several **range** scans and to merge their results into one. The merge can produce unions, intersections, or unions-of-intersections of its underlying scans.

In **EXPLAIN** output, the Index Merge method appears as **index_merge** in the **type** column. In this case, the **key** column contains a list of indexes used, and **key_len** contains a list of the longest key parts for those indexes.

Examples:

```
SELECT * FROM tbl_name WHERE key1 = 10 OR key2 = 20;

SELECT * FROM tbl_name
WHERE (key1 = 10 OR key2 = 20) AND non_key=30;

SELECT * FROM t1, t2
WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
AND t2.key1=t1.some_col;

SELECT * FROM t1, t2
WHERE t1.key1=1
AND (t2.key1=t1.some_col OR t2.key2=t1.some_col2);
```

The Index Merge method has several access algorithms (seen in the **Extra** field of **EXPLAIN** output):

- Using **intersect(...)**
- Using **union(...)**
- Using **sort_union(...)**

The following sections describe these methods in greater detail.

Note

The Index Merge optimization algorithm has the following known deficiencies:

- If a range scan is possible on some key, the optimizer will not consider using Index Merge Union or Index Merge Sort-Union algorithms. For example, consider this query:

```
SELECT * FROM t1 WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
```

For this query, two plans are possible:

- An Index Merge scan using the `(goodkey1 < 10 OR goodkey2 < 20)` condition.
- A range scan using the `badkey < 30` condition.

However, the optimizer considers only the second plan.

- If your query has a complex `WHERE` clause with deep `AND/OR` nesting and MySQL doesn't choose the optimal plan, try distributing terms using the following identity laws:

```
(x AND y) OR z = (x OR z) AND (y OR z)
(x OR y) AND z = (x AND z) OR (y AND z)
```

- Index Merge is not applicable to fulltext indexes. We plan to extend it to cover these in a future MySQL release.

The choice between different possible variants of the Index Merge access method and other access methods is based on cost estimates of various available options.

6.2.6.1. The Index Merge Intersection Access Algorithm

This access algorithm can be employed when a `WHERE` clause was converted to several range conditions on different keys combined with `AND`, and each condition is one of the following:

- In this form, where the index has exactly `N` parts (that is, all index parts are covered):

```
key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
```

- Any range condition over a primary key of an `InnoDB` table.

Examples:

```
SELECT * FROM innodb_table WHERE primary_key < 10 AND key_col1=20;
SELECT * FROM tbl_name
WHERE (key1_part1=1 AND key1_part2=2) AND key2=2;
```

The Index Merge intersection algorithm performs simultaneous scans on all used indexes and produces the intersection of row sequences that it receives from the merged index scans.

If all columns used in the query are covered by the used indexes, full table rows are not retrieved (`EXPLAIN` output contains `Using index` in `Extra` field in this case). Here is an example of such a query:

```
SELECT COUNT(*) FROM t1 WHERE key1=1 AND key2=1;
```

If the used indexes don't cover all columns used in the query, full rows are retrieved only when the range conditions for all used keys are satisfied.

If one of the merged conditions is a condition over a primary key of an `InnoDB` table, it is not used for row retrieval, but is used to filter out rows retrieved using other conditions.

6.2.6.2. The Index Merge Union Access Algorithm

The applicability criteria for this algorithm are similar to those for the Index Merge method intersection algorithm. The algorithm can be employed when the table's `WHERE` clause was converted to several range conditions on different keys combined with `OR`, and each condition is one of the following:

- In this form, where the index has exactly *N* parts (that is, all index parts are covered):

```
key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
```

- Any range condition over a primary key of an **InnoDB** table.
- A condition for which the Index Merge method intersection algorithm is applicable.

Examples:

```
SELECT * FROM t1 WHERE key1=1 OR key2=2 OR key3=3;
SELECT * FROM innodb_table WHERE (key1=1 AND key2=2) OR
(key3='foo' AND key4='bar') AND key5=5;
```

6.2.6.3. The Index Merge Sort-Union Access Algorithm

This access algorithm is employed when the **WHERE** clause was converted to several range conditions combined by **OR**, but for which the Index Merge method union algorithm is not applicable.

Examples:

```
SELECT * FROM tbl_name WHERE key_col1 < 10 OR key_col2 < 20;
SELECT * FROM tbl_name
WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col=30;
```

The difference between the sort-union algorithm and the union algorithm is that the sort-union algorithm must first fetch row IDs for all rows and sort them before returning any rows.

6.2.7. IS NULL Optimization

MySQL can perform the same optimization on *col_name IS NULL* that it can use for *col_name = constant_value*. For example, MySQL can use indexes and ranges to search for **NULL** with **IS NULL**.

Examples:

```
SELECT * FROM tbl_name WHERE key_col IS NULL;
SELECT * FROM tbl_name WHERE key_col <=> NULL;
SELECT * FROM tbl_name
WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

If a **WHERE** clause includes a *col_name IS NULL* condition for a column that is declared as **NOT NULL**, that expression is optimized away. This optimization does not occur in cases when the column might produce **NULL** anyway; for example, if it comes from a table on the right side of a **LEFT JOIN**.

MySQL can also optimize the combination *col_name = expr OR col_name IS NULL*, a form that is common in resolved subqueries. **EXPLAIN** shows **ref_or_null** when this optimization is used.

This optimization can handle one **IS NULL** for any key part.

Some examples of queries that are optimized, assuming that there is an index on columns **a** and **b** of table **t2**:

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;
SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;
SELECT * FROM t1, t2
WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;
SELECT * FROM t1, t2
WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);
SELECT * FROM t1, t2
WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)
OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

`ref_or_null` works by first doing a read on the reference key, and then a separate search for rows with a `NULL` key value.

Note that the optimization can handle only one `IS NULL` level. In the following query, MySQL uses key lookups only on the expression `(t1.a=t2.a AND t2.a IS NULL)` and is not able to use the key part on `b`:

```
SELECT * FROM t1, t2
WHERE (t1.a=t2.a AND t2.a IS NULL)
OR (t1.b=t2.b AND t2.b IS NULL);
```

6.2.8. LEFT JOIN and RIGHT JOIN Optimization

MySQL implements an `A LEFT JOIN B join_condition` as follows:

- Table `B` is set to depend on table `A` and all tables on which `A` depends.
- Table `A` is set to depend on all tables (except `B`) that are used in the `LEFT JOIN` condition.
- The `LEFT JOIN` condition is used to decide how to retrieve rows from table `B`. (In other words, any condition in the `WHERE` clause is not used.)
- All standard join optimizations are performed, with the exception that a table is always read after all tables on which it depends. If there is a circular dependence, MySQL issues an error.
- All standard `WHERE` optimizations are performed.
- If there is a row in `A` that matches the `WHERE` clause, but there is no row in `B` that matches the `ON` condition, an extra `B` row is generated with all columns set to `NULL`.
- If you use `LEFT JOIN` to find rows that do not exist in some table and you have the following test: `col_name IS NULL` in the `WHERE` part, where `col_name` is a column that is declared as `NOT NULL`, MySQL stops searching for more rows (for a particular key combination) after it has found one row that matches the `LEFT JOIN` condition.

The implementation of `RIGHT JOIN` is analogous to that of `LEFT JOIN` with the roles of the tables reversed.

The join optimizer calculates the order in which tables should be joined. The table read order forced by `LEFT JOIN` or `STRAIGHT_JOIN` helps the join optimizer do its work much more quickly, because there are fewer table permutations to check. Note that this means that if you do a query of the following type, MySQL does a full scan on `b` because the `LEFT JOIN` forces it to be read before `d`:

```
SELECT *
FROM a JOIN b LEFT JOIN c ON (c.key=a.key)
LEFT JOIN d ON (d.key=a.key)
WHERE b.key=d.key;
```

The fix in this case is reverse the order in which `a` and `b` are listed in the `FROM` clause:

```
SELECT *
FROM b JOIN a LEFT JOIN c ON (c.key=a.key)
LEFT JOIN d ON (d.key=a.key)
WHERE b.key=d.key;
```

For a `LEFT JOIN`, if the `WHERE` condition is always false for the generated `NULL` row, the `LEFT JOIN` is changed to a normal join. For example, the `WHERE` clause would be false in the following query if `t2.column1` were `NULL`:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

Therefore, it is safe to convert the query to a normal join:

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

This can be made faster because MySQL can use table `t2` before table `t1` if doing so would result in a better query plan. To force a specific table order, use `STRAIGHT_JOIN`.

6.2.9. Nested Join Optimization

The syntax for expressing joins allows nested joins. The following discussion refers to the join syntax described in [Section 12.2.7.1, “JOIN Syntax”](#).

The syntax of *table_factor* is extended in comparison with the SQL Standard. The latter accepts only *table_reference*, not a list of them inside a pair of parentheses. This is a conservative extension if we consider each comma in a list of *table_reference* items as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
      ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
      ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

In MySQL, **CROSS JOIN** is a syntactic equivalent to **INNER JOIN** (they can replace each other). In standard SQL, they are not equivalent. **INNER JOIN** is used with an **ON** clause; **CROSS JOIN** is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. After removing parentheses and grouping operations to the left, the join expression:

```
t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
      ON t1.a=t2.a
```

transforms into the expression:

```
(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3
      ON t2.b=t3.b OR t2.b IS NULL
```

Yet, the two expressions are not equivalent. To see this, suppose that the tables **t1**, **t2**, and **t3** have the following state:

- Table **t1** contains rows (1), (2)
- Table **t2** contains row (1, 101)
- Table **t3** contains row (101)

In this case, the first expression returns a result set including the rows (1, 1, 101, 101), (2, NULL, NULL, NULL), whereas the second expression returns the rows (1, 1, 101, 101), (2, NULL, NULL, 101):

```
mysql> SELECT *
-> FROM t1
-> LEFT JOIN
-> (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
-> ON t1.a=t2.a;
```

a	a	b	b
1	1	101	101
2	NULL	NULL	NULL

```
mysql> SELECT *
-> FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)
-> LEFT JOIN t3
-> ON t2.b=t3.b OR t2.b IS NULL;
```

a	a	b	b
1	1	101	101
2	NULL	NULL	101

In the following example, an outer join operation is used together with an inner join operation:

```
t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```


That expression cannot be transformed into the following expression:

```
t1 LEFT JOIN t2 ON t1.a=t2.a, t3.
```

For the given table states, the two expressions return different sets of rows:

```
mysql> SELECT *
-> FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;
+-----+-----+-----+-----+
| a | a | b | b |
+-----+-----+-----+-----+
| 1 | 1 | 101 | 101 |
| 2 | NULL | NULL | NULL |
+-----+-----+-----+-----+

mysql> SELECT *
-> FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;
+-----+-----+-----+-----+
| a | a | b | b |
+-----+-----+-----+-----+
| 1 | 1 | 101 | 101 |
| 2 | NULL | NULL | 101 |
+-----+-----+-----+-----+
```

Therefore, if we omit parentheses in a join expression with outer join operators, we might change the result set for the original expression.

More exactly, we cannot ignore parentheses in the right operand of the left outer join operation and in the left operand of a right join operation. In other words, we cannot ignore parentheses for the inner table expressions of outer join operations. Parentheses for the other operand (operand for the outer table) can be ignored.

The following expression:

```
(t1,t2) LEFT JOIN t3 ON P(t2.b,t3.b)
```

is equivalent to this expression:

```
t1, t2 LEFT JOIN t3 ON P(t2.b,t3.b)
```

for any tables **t1**, **t2**, **t3** and any condition **P** over attributes **t2.b** and **t3.b**.

Whenever the order of execution of the join operations in a join expression (*join_table*) is not from left to right, we talk about nested joins. Consider the following queries:

```
SELECT * FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a
WHERE t1.a > 1

SELECT * FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1
```

Those queries are considered to contain these nested joins:

```
t2 LEFT JOIN t3 ON t2.b=t3.b
t2, t3
```

The nested join is formed in the first query with a left join operation, whereas in the second query it is formed with an inner join operation.

In the first query, the parentheses can be omitted: The grammatical structure of the join expression will dictate the same order of execution for join operations. For the second query, the parentheses cannot be omitted, although the join expression here can be interpreted unambiguously without them. (In our extended syntax the parentheses in **(t2, t3)** of the second query are required, although theoretically the query could be parsed without them: We still would have unambiguous syntactical structure for the query because **LEFT JOIN** and **ON** would play the role of the left and right delimiters for the expression **(t2,t3)**.)

The preceding examples demonstrate these points:

- For join expressions involving only inner joins (and not outer joins), parentheses can be removed. You can remove parentheses and evaluate left to right (or, in fact, you can evaluate the tables in any order).

- The same is not true, in general, for outer joins or for outer joins mixed with inner joins. Removal of parentheses may change the result.

Queries with nested outer joins are executed in the same pipeline manner as queries with inner joins. More exactly, a variation of the nested-loop join algorithm is exploited. Recall by what algorithmic schema the nested-loop join executes a query. Suppose that we have a join query over 3 tables **T1**, **T2**, **T3** of the form:

```
SELECT * FROM T1 INNER JOIN T2 ON P1(T1,T2)
           INNER JOIN T3 ON P2(T2,T3)
WHERE P(T1,T2,T3).
```

Here, **P1(T1, T2)** and **P2(T2, T3)** are some join conditions (on expressions), whereas **P(t1, t2, t3)** is a condition over columns of tables **T1**, **T2**, **T3**.

The nested-loop join algorithm would execute this query in the following manner:

```
FOR each row t1 in T1 {
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

The notation **t1 || t2 || t3** means “a row constructed by concatenating the columns of rows **t1**, **t2**, and **t3**.” In some of the following examples, **NULL** where a row name appears means that **NULL** is used for each column of that row. For example, **t1 || t2 || NULL** means “a row constructed by concatenating the columns of rows **t1** and **t2**, and **NULL** for each column of **t3**.”

Now let's consider a query with nested outer joins:

```
SELECT * FROM T1 LEFT JOIN
           (T2 LEFT JOIN T3 ON P2(T2,T3))
           ON P1(T1,T2)
WHERE P(T1,T2,T3).
```

For this query, we modify the nested-loop pattern to get:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
    }
    f1=TRUE;
  }
  IF (!f2) {
    IF P(t1,t2,NULL) {
      t:=t1||t2||NULL; OUTPUT t;
    }
    f1=TRUE;
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

In general, for any nested loop for the first inner table in an outer join operation, a flag is introduced that is turned off before the loop and is checked after the loop. The flag is turned on when for the current row from the outer table a match from the table representing the inner operand is found. If at the end of the loop cycle the flag is still off, no match has been found for the current row of the outer table. In this case, the row is complemented by **NULL** values for the columns of the inner tables. The result row is passed to the final check for the output or into the next nested loop, but only if the row satisfies the join condition of all embedded outer joins.

In our example, the outer join table expressed by the following expression is embedded:

```
(T2 LEFT JOIN T3 ON P2(T2,T3))
```

Note that for the query with inner joins, the optimizer could choose a different order of nested loops, such as this one:

```
FOR each row t3 in T3 {
  FOR each row t2 in T2 such that P2(t2,t3) {
    FOR each row t1 in T1 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

For the queries with outer joins, the optimizer can choose only such an order where loops for outer tables precede loops for inner tables. Thus, for our query with outer joins, only one nesting order is possible. For the following query, the optimizer will evaluate two different nestings:

```
SELECT * T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)
WHERE P(T1,T2,T3)
```

The nestings are these:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t1,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
    f1:=TRUE
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

and:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t3 in T3 such that P2(t1,t3) {
    FOR each row t2 in T2 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
    f1:=TRUE
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

In both nestings, **T1** must be processed in the outer loop because it is used in an outer join. **T2** and **T3** are used in an inner join, so that join must be processed in the inner loop. However, because the join is an inner join, **T2** and **T3** can be processed in either order.

When discussing the nested-loop algorithm for inner joins, we omitted some details whose impact on the performance of query execution may be huge. We did not mention so-called “pushed-down” conditions. Suppose that our **WHERE** condition **P(T1, T2, T3)** can be represented by a conjunctive formula:

```
P(T1,T2,T2) = C1(T1) AND C2(T2) AND C3(T3).
```

In this case, MySQL actually uses the following nested-loop schema for the execution of the query with inner joins:

```
FOR each row t1 in T1 such that C1(t1) {
  FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2) {
    FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

```

}
}
}

```

You see that each of the conjuncts $C1(T1)$, $C2(T2)$, $C3(T3)$ are pushed out of the most inner loop to the most outer loop where it can be evaluated. If $C1(T1)$ is a very restrictive condition, this condition pushdown may greatly reduce the number of rows from table $T1$ passed to the inner loops. As a result, the execution time for the query may improve immensely.

For a query with outer joins, the **WHERE** condition is to be checked only after it has been found that the current row from the outer table has a match in the inner tables. Thus, the optimization of pushing conditions out of the inner nested loops cannot be applied directly to queries with outer joins. Here we have to introduce conditional pushed-down predicates guarded by the flags that are turned on when a match has been encountered.

For our example with outer joins with:

```
P(T1,T2,T3)=C1(T1) AND C(T2) AND C3(T3)
```

the nested-loop schema using guarded pushed-down conditions looks like this:

```

FOR each row t1 in T1 such that C1(t1) {
  BOOL f1:=FALSE;
  FOR each row t2 in T2
    such that P1(t1,t2) AND (f1?C2(t2):TRUE) {
      BOOL f2:=FALSE;
      FOR each row t3 in T3
        such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {
          IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {
            t:=t1||t2||t3; OUTPUT t;
          }
          f2=TRUE;
          f1=TRUE;
        }
      }
    IF (!f2) {
      IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
}
IF (!f1 && P(t1,NULL,NULL)) {
  t:=t1||NULL||NULL; OUTPUT t;
}
}

```

In general, pushed-down predicates can be extracted from join conditions such as $P1(T1, T2)$ and $P(T2, T3)$. In this case, a pushed-down predicate is guarded also by a flag that prevents checking the predicate for the **NULL**-complemented row generated by the corresponding outer join operation.

Note that access by key from one inner table to another in the same nested join is prohibited if it is induced by a predicate from the **WHERE** condition. (We could use conditional key access in this case, but this technique is not employed yet in MySQL 5.1.)

6.2.10. Outer Join Simplification

Table expressions in the **FROM** clause of a query are simplified in many cases.

At the parser stage, queries with right outer joins operations are converted to equivalent queries containing only left join operations. In the general case, the conversion is performed according to the following rule:

```

(T1, ...) RIGHT JOIN (T2,...) ON P(T1,...,T2,...) =
(T2, ...) LEFT JOIN (T1,...) ON P(T1,...,T2,...)

```

All inner join expressions of the form $T1 \text{ INNER JOIN } T2 \text{ ON } P(T1, T2)$ are replaced by the list $T1, T2, P(T1, T2)$ being joined as a conjunct to the **WHERE** condition (or to the join condition of the embedding join, if there is any).

When the optimizer evaluates plans for join queries with outer join operation, it takes into consideration only the plans where, for each such operation, the outer tables are accessed before the inner tables. The optimizer options are limited because only such plans enables us to execute queries with outer joins operations by the nested loop schema.

Suppose that we have a query of the form:

```
SELECT * T1 LEFT JOIN T2 ON P1(T1,T2)
```

```
WHERE P(T1, T2) AND R(T2)
```

with **R(T2)** narrowing greatly the number of matching rows from table **T2**. If we executed the query as it is, the optimizer would have no other choice besides to access table **T1** before table **T2** that may lead to a very inefficient execution plan.

Fortunately, MySQL converts such a query into a query without an outer join operation if the **WHERE** condition is null-rejected. A condition is called null-rejected for an outer join operation if it evaluates to **FALSE** or to **UNKNOWN** for any **NULL**-complemented row built for the operation.

Thus, for this outer join:

```
T1 LEFT JOIN T2 ON T1.A=T2.A
```

Conditions such as these are null-rejected:

```
T2.B IS NOT NULL,
T2.B > 3,
T2.C <= T1.C,
T2.B < 2 OR T2.C > 1
```

Conditions such as these are not null-rejected:

```
T2.B IS NULL,
T1.B < 3 OR T2.B IS NOT NULL,
T1.B < 3 OR T2.B > 3
```

The general rules for checking whether a condition is null-rejected for an outer join operation are simple. A condition is null-rejected in the following cases:

- If it is of the form **A IS NOT NULL**, where **A** is an attribute of any of the inner tables
- If it is a predicate containing a reference to an inner table that evaluates to **UNKNOWN** when one of its arguments is **NULL**
- If it is a conjunction containing a null-rejected condition as a conjunct
- If it is a disjunction of null-rejected conditions

A condition can be null-rejected for one outer join operation in a query and not null-rejected for another. In the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                LEFT JOIN T3 ON T3.B=T1.B
WHERE T3.C > 0
```

the **WHERE** condition is null-rejected for the second outer join operation but is not null-rejected for the first one.

If the **WHERE** condition is null-rejected for an outer join operation in a query, the outer join operation is replaced by an inner join operation.

For example, the preceding query is replaced with the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                INNER JOIN T3 ON T3.B=T1.B
WHERE T3.C > 0
```

For the original query, the optimizer would evaluate plans compatible with only one access order **T1, T2, T3**. For the replacing query, it additionally considers the access sequence **T3, T1, T2**.

A conversion of one outer join operation may trigger a conversion of another. Thus, the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                LEFT JOIN T3 ON T3.B=T2.B
WHERE T3.C > 0
```

will be first converted to the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
      INNER JOIN T3 ON T3.B=T2.B
WHERE T3.C > 0
```

which is equivalent to the query:

```
SELECT * FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3
WHERE T3.C > 0 AND T3.B=T2.B
```

Now the remaining outer join operation can be replaced by an inner join, too, because the condition `T3.B=T2.B` is null-rejected and we get a query without outer joins at all:

```
SELECT * FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3
WHERE T3.C > 0 AND T3.B=T2.B
```

Sometimes we succeed in replacing an embedded outer join operation, but cannot convert the embedding outer join. The following query:

```
SELECT * FROM T1 LEFT JOIN
      (T2 LEFT JOIN T3 ON T3.B=T2.B)
      ON T2.A=T1.A
WHERE T3.C > 0
```

is converted to:

```
SELECT * FROM T1 LEFT JOIN
      (T2 INNER JOIN T3 ON T3.B=T2.B)
      ON T2.A=T1.A
WHERE T3.C > 0,
```

That can be rewritten only to the form still containing the embedding outer join operation:

```
SELECT * FROM T1 LEFT JOIN
      (T2, T3)
      ON (T2.A=T1.A AND T3.B=T2.B)
WHERE T3.C > 0.
```

When trying to convert an embedded outer join operation in a query, we must take into account the join condition for the embedding outer join together with the `WHERE` condition. In the query:

```
SELECT * FROM T1 LEFT JOIN
      (T2 LEFT JOIN T3 ON T3.B=T2.B)
      ON T2.A=T1.A AND T3.C=T1.C
WHERE T3.D > 0 OR T1.D > 0
```

the `WHERE` condition is not null-rejected for the embedded outer join, but the join condition of the embedding outer join `T2.A=T1.A AND T3.C=T1.C` is null-rejected. So the query can be converted to:

```
SELECT * FROM T1 LEFT JOIN
      (T2, T3)
      ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B
WHERE T3.D > 0 OR T1.D > 0
```

6.2.11. ORDER BY Optimization

In some cases, MySQL can use an index to satisfy an `ORDER BY` clause without doing any extra sorting.

The index can also be used even if the `ORDER BY` does not match the index exactly, as long as all of the unused portions of the index and all the extra `ORDER BY` columns are constants in the `WHERE` clause. The following queries use the index to resolve the `ORDER BY` part:

```
SELECT * FROM t1
ORDER BY key_part1, key_part2, ... ;

SELECT * FROM t1
WHERE key_part1=constant
ORDER BY key_part2;

SELECT * FROM t1
```

```
ORDER BY key_part1 DESC, key_part2 DESC;
SELECT * FROM t1
WHERE key_part1=1
ORDER BY key_part1 DESC, key_part2 DESC;
```

In some cases, MySQL *cannot* use indexes to resolve the **ORDER BY**, although it still uses indexes to find the rows that match the **WHERE** clause. These cases include the following:

- You use **ORDER BY** on different keys:

```
SELECT * FROM t1 ORDER BY key1, key2;
```

- You use **ORDER BY** on non-consecutive parts of a key:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;
```

- You mix **ASC** and **DESC**:

```
SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
```

- The key used to fetch the rows is not the same as the one used in the **ORDER BY**:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
```

- You are joining many tables, and the columns in the **ORDER BY** are not all from the first non-constant table that is used to retrieve rows. (This is the first table in the **EXPLAIN** output that does not have a **const** join type.)
- You have different **ORDER BY** and **GROUP BY** expressions.
- The type of table index used does not store rows in order. For example, this is true for a **HASH** index in a **MEMORY** table.

With **EXPLAIN SELECT ... ORDER BY**, you can check whether MySQL can use indexes to resolve the query. It cannot if you see **Using filesort** in the **Extra** column. See [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#).

MySQL has two **filesort** algorithms for sorting and retrieving results. The original method uses only the **ORDER BY** columns. The modified method uses not just the **ORDER BY** columns, but all the columns used in the query.

The optimizer selects which **filesort** algorithm to use. It normally uses the modified algorithm except when **BLOB** or **TEXT** columns are involved, in which case it uses the original algorithm.

The original **filesort** algorithm works as follows:

- Read all rows according to key or by table scanning. Rows that do not match the **WHERE** clause are skipped.
- For each row, store a pair of values in a buffer (the sort key and the row pointer). The size of the buffer is the value of the **sort_buffer_size** system variable.
- When the buffer gets full, run a qsort (quicksort) on it and store the result in a temporary file. Save a pointer to the sorted block. (If all pairs fit into the sort buffer, no temporary file is created.)
- Repeat the preceding steps until all rows have been read.
- Do a multi-merge of up to **MERGEBUFF** (7) regions to one block in another temporary file. Repeat until all blocks from the first file are in the second file.
- Repeat the following until there are fewer than **MERGEBUFF2** (15) blocks left.
- On the last multi-merge, only the pointer to the row (the last part of the sort key) is written to a result file.
- Read the rows in sorted order by using the row pointers in the result file. To optimize this, we read in a big block of row pointers, sort them, and use them to read the rows in sorted order into a row buffer. The size of the buffer is the value of the **read_rnd_buffer_size** system variable. The code for this step is in the [sql/records.cc](#) source file.

One problem with this approach is that it reads rows twice: One time when evaluating the `WHERE` clause, and again after sorting the pair values. And even if the rows were accessed successively the first time (for example, if a table scan is done), the second time they are accessed randomly. (The sort keys are ordered, but the row positions are not.)

The modified `filesort` algorithm incorporates an optimization such that it records not only the sort key value and row position, but also the columns required for the query. This avoids reading the rows twice. The modified `filesort` algorithm works like this:

1. Read the rows that match the `WHERE` clause.
2. For each row, record a tuple of values consisting of the sort key value and row position, and also the columns required for the query.
3. Sort the tuples by sort key value
4. Retrieve the rows in sorted order, but read the required columns directly from the sorted tuples rather than by accessing the table a second time.

Using the modified `filesort` algorithm, the tuples are longer than the pairs used in the original method, and fewer of them fit in the sort buffer (the size of which is given by `sort_buffer_size`). As a result, it is possible for the extra I/O to make the modified approach slower, not faster. To avoid a slowdown, the optimization is used only if the total size of the extra columns in the sort tuple does not exceed the value of the `max_length_for_sort_data` system variable. (A symptom of setting the value of this variable too high is that you should see high disk activity and low CPU activity.)

For slow queries for which `filesort` is not used, you might try lowering `max_length_for_sort_data` to a value that is appropriate to trigger a `filesort`.

If you want to increase `ORDER BY` speed, check whether you can get MySQL to use indexes rather than an extra sorting phase. If this is not possible, you can try the following strategies:

- Increase the size of the `sort_buffer_size` variable.
- Increase the size of the `read_rnd_buffer_size` variable.
- Use less RAM per row by declaring columns only as large as they need to be to hold the values stored in them. For example, `CHAR(16)` is better than `CHAR(200)` if values never exceed 16 characters.
- Change `tmpdir` to point to a dedicated filesystem with large amounts of free space. Also, this option accepts several paths that are used in round-robin fashion, so you can use this feature to spread the load across several directories. Paths should be separated by colon characters (':') on Unix and semicolon characters(';') on Windows, NetWare, and OS/2. *Note:* The paths should be for directories in filesystems that are located on different *physical* disks, not different partitions on the same disk.

By default, MySQL sorts all `GROUP BY col1, col2, ...` queries as if you specified `ORDER BY col1, col2, ...` in the query as well. If you include an `ORDER BY` clause explicitly that contains the same column list, MySQL optimizes it away without any speed penalty, although the sorting still occurs. If a query includes `GROUP BY` but you want to avoid the overhead of sorting the result, you can suppress sorting by specifying `ORDER BY NULL`. For example:

```
INSERT INTO foo
SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

6.2.12. GROUP BY Optimization

The most general way to satisfy a `GROUP BY` clause is to scan the whole table and create a new temporary table where all rows from each group are consecutive, and then use this temporary table to discover groups and apply aggregate functions (if any). In some cases, MySQL is able to do much better than that and to avoid creation of temporary tables by using index access.

The most important preconditions for using indexes for `GROUP BY` are that all `GROUP BY` columns reference attributes from the same index, and that the index stores its keys in order (for example, this is a `BTREE` index and not a `HASH` index). Whether use of temporary tables can be replaced by index access also depends on which parts of an index are used in a query, the conditions specified for these parts, and the selected aggregate functions.

In MySQL, `GROUP BY` is used for sorting, so the server may also apply `ORDER BY` optimizations to grouping. See [Section 6.2.11](#),

ORDER BY Optimization”.

There are two ways to execute a **GROUP BY** query via index access, as detailed in the following sections. In the first method, the grouping operation is applied together with all range predicates (if any). The second method first performs a range scan, and then groups the resulting tuples.

6.2.12.1. Loose index scan

The most efficient way to process **GROUP BY** is when the index is used to directly retrieve the group fields. With this access method, MySQL uses the property of some index types that the keys are ordered (for example, **BTREE**). This property enables use of lookup groups in an index without having to consider all keys in the index that satisfy all **WHERE** conditions. This access method considers only a fraction of the keys in an index, so it is called a *loose index scan*. When there is no **WHERE** clause, a loose index scan reads as many keys as the number of groups, which may be a much smaller number than that of all keys. If the **WHERE** clause contains range predicates (see the discussion of the **range** join type in [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#)), a loose index scan looks up the first key of each group that satisfies the range conditions, and again reads the least possible number of keys. This is possible under the following conditions:

- The query is over a single table.
- The **GROUP BY** includes the first consecutive parts of the index. (If, instead of **GROUP BY**, the query has a **DISTINCT** clause, all distinct attributes refer to the beginning of the index.)
- The only aggregate functions used (if any) are **MIN()** and **MAX()**, and all of them refer to the same column.
- Any other parts of the index than those from the **GROUP BY** referenced in the query must be constants (that is, they must be referenced in equalities with constants), except for the argument of **MIN()** or **MAX()** functions.

The **EXPLAIN** output for such queries shows **Using index for group-by** in the **Extra** column.

The following queries fall into this category, assuming that there is an index **idx(c1,c2,c3)** on table **t1(c1,c2,c3,c4)**:

```
SELECT c1, c2 FROM t1 GROUP BY c1, c2;
SELECT DISTINCT c1, c2 FROM t1;
SELECT c1, MIN(c2) FROM t1 GROUP BY c1;
SELECT c1, c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > const GROUP BY c1, c2;
SELECT c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT c1, c2 FROM t1 WHERE c3 = const GROUP BY c1, c2;
```

The following queries cannot be executed with this quick select method, for the reasons given:

- There are aggregate functions other than **MIN()** or **MAX()**, for example:

```
SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
```

- The fields in the **GROUP BY** clause do not refer to the beginning of the index, as shown here:

```
SELECT c1,c2 FROM t1 GROUP BY c2, c3;
```

- The query refers to a part of a key that comes after the **GROUP BY** part, and for which there is no equality with a constant, an example being:

```
SELECT c1,c3 FROM t1 GROUP BY c1, c2;
```

6.2.12.2. Tight index scan

A tight index scan may be either a full index scan or a range index scan, depending on the query conditions.

When the conditions for a loose index scan are not met, it is still possible to avoid creation of temporary tables for **GROUP BY** queries. If there are range conditions in the **WHERE** clause, this method reads only the keys that satisfy these conditions. Otherwise, it performs an index scan. Because this method reads all keys in each range defined by the **WHERE** clause, or scans the whole index if there are no range conditions, we term it a *tight index scan*. Notice that with a tight index scan, the grouping operation is performed only after all

keys that satisfy the range conditions have been found.

For this method to work, it is sufficient that there is a constant equality condition for all columns in a query referring to parts of the key coming before or in between parts of the **GROUP BY** key. The constants from the equality conditions fill in any “gaps” in the search keys so that it is possible to form complete prefixes of the index. These index prefixes then can be used for index lookups. If we require sorting of the **GROUP BY** result, and it is possible to form search keys that are prefixes of the index, MySQL also avoids extra sorting operations because searching with prefixes in an ordered index already retrieves all the keys in order.

The following queries do not work with the loose index scan access method described earlier, but still work with the tight index scan access method (assuming that there is an index `idx(c1, c2, c3)` on table `t1(c1, c2, c3, c4)`).

- There is a gap in the **GROUP BY**, but it is covered by the condition `c2 = 'a'`:

```
SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
```

- The **GROUP BY** does not begin with the first part of the key, but there is a condition that provides a constant for that part:

```
SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
```

6.2.13. DISTINCT Optimization

DISTINCT combined with **ORDER BY** needs a temporary table in many cases.

Because **DISTINCT** may use **GROUP BY**, you should be aware of how MySQL works with columns in **ORDER BY** or **HAVING** clauses that are not part of the selected columns. See [Section 11.12.3, “GROUP BY and HAVING with Hidden Fields”](#).

In most cases, a **DISTINCT** clause can be considered as a special case of **GROUP BY**. For example, the following two queries are equivalent:

```
SELECT DISTINCT c1, c2, c3 FROM t1
WHERE c1 > const;

SELECT c1, c2, c3 FROM t1
WHERE c1 > const GROUP BY c1, c2, c3;
```

Due to this equivalence, the optimizations applicable to **GROUP BY** queries can be also applied to queries with a **DISTINCT** clause. Thus, for more details on the optimization possibilities for **DISTINCT** queries, see [Section 6.2.12, “GROUP BY Optimization”](#).

When combining **LIMIT row_count** with **DISTINCT**, MySQL stops as soon as it finds *row_count* unique rows.

If you do not use columns from all tables named in a query, MySQL stops scanning any unused tables as soon as it finds the first match. In the following case, assuming that `t1` is used before `t2` (which you can check with **EXPLAIN**), MySQL stops reading from `t2` (for any particular row in `t1`) when it finds the first row in `t2`:

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

6.2.14. Optimizing IN/=ANY Subqueries

Certain optimizations are applicable to comparisons that use the **IN** operator to test subquery results (or that use **=ANY**, which is equivalent). This section discusses these optimizations, particularly with regard to the challenges that **NULL** values present. Suggestions on what you can do to help the optimizer are given at the end of the discussion.

Consider the following subquery comparison:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

MySQL evaluates queries “from outside to inside.” That is, it first obtains the value of the outer expression *outer_expr*, and then runs the subquery and captures the rows that it produces.

A very useful optimization is to “inform” the subquery that the only rows of interest are those where the inner expression *inner_expr* is equal to *outer_expr*. This is done by pushing down an appropriate equality into the subquery’s **WHERE** clause. That is, the comparison is converted to this:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

After the conversion, MySQL can use the pushed-down equality to limit the number of rows that it must examine when evaluating the subquery.

More generally, a comparison of *N* values to a subquery that returns *N*-value rows is subject to the same conversion. If *oe_i* and *ie_i* represent corresponding outer and inner expression values, this subquery comparison:

```
(oe_1, ..., oe_N) IN  
(SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

Becomes:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where  
AND oe_1 = ie_1  
AND ...  
AND oe_N = ie_N)
```

The following discussion assumes a single pair of outer and inner expression values for simplicity.

The conversion just described has its limitations. It is valid only if we ignore possible **NULL** values. That is, the “pushdown” strategy works as long as both of these two conditions are true:

- *outer_expr* and *inner_expr* cannot be **NULL**.
- You do not need to distinguish **NULL** from **FALSE** subquery results. (If the subquery is a part of an **OR** or **AND** expression in the **WHERE** clause, MySQL assumes that you don't care.)

When either or both of those conditions do not hold, optimization is more complex.

Suppose that *outer_expr* is known to be a non-**NULL** value but the subquery does not produce a row such that *outer_expr* = *inner_expr*. Then *outer_expr* IN (SELECT ...) evaluates as follows:

- **NULL**, if the **SELECT** produces any row where *inner_expr* is **NULL**
- **FALSE**, if the **SELECT** produces only non-**NULL** values or produces nothing

In this situation, the approach of looking for rows with *outer_expr* = *inner_expr* is no longer valid. It is necessary to look for such rows, but if none are found, also look for rows where *inner_expr* is **NULL**. Roughly speaking, the subquery can be converted to:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND  
(outer_expr=inner_expr OR inner_expr IS NULL))
```

The need to evaluate the extra **IS NULL** condition is why MySQL has the *ref_or_null* access method:

```
mysql> EXPLAIN  
-> SELECT outer_expr IN (SELECT t2.maybe_null_key  
-> FROM t2, t3 WHERE ...)  
-> FROM t1;  
***** 1. row *****  
id: 1  
select_type: PRIMARY  
table: t1  
...  
***** 2. row *****  
id: 2  
select_type: DEPENDENT SUBQUERY  
table: t2  
type: ref_or_null  
possible_keys: maybe_null_key  
key: maybe_null_key  
key_len: 5  
ref: func  
rows: 2  
Extra: Using where; Using index  
...
```

The `unique_subquery` and `index_subquery` subquery-specific access methods also have or-null variants. However, they are not visible in `EXPLAIN` output, so you must use `EXPLAIN EXTENDED` followed by `SHOW WARNINGS` (note the `checking NULL` in the warning message):

```
mysql> EXPLAIN EXTENDED
-> SELECT outer_expr IN (SELECT maybe_null_key FROM t2) FROM t1\G
***** 1. row *****
      id: 1
    select_type: PRIMARY
      table: t1
    ...
***** 2. row *****
      id: 2
    select_type: DEPENDENT SUBQUERY
      table: t2
        type: index_subquery
possible_keys: maybe_null_key
          key: maybe_null_key
        key_len: 5
          ref: func
         rows: 2
        Extra: Using index

mysql> SHOW WARNINGS\G
***** 1. row *****
    Level: Note
     Code: 1003
Message: select (`test`.`t1`.`outer_expr`,
               (((`test`.`t1`.`outer_expr`) in t2 on
                maybe_null_key checking NULL))) AS `outer_expr` IN (SELECT
                maybe_null_key FROM t2)` from `test`.`t1`
```

The additional `OR ... IS NULL` condition makes query execution slightly more complicated (and some optimizations within the subquery become inapplicable), but generally this is tolerable.

The situation is much worse when `outer_expr` can be `NULL`. According to the SQL interpretation of `NULL` as “unknown value,” `NULL IN (SELECT inner_expr ...)` should evaluate to:

- `NULL`, if the `SELECT` produces any rows
- `FALSE`, if the `SELECT` produces no rows

For proper evaluation, it is necessary to be able to check whether the `SELECT` has produced any rows at all, so `outer_expr = inner_expr` cannot be pushed down into the subquery. This is a problem, because many real world subqueries become very slow unless the equality can be pushed down.

Essentially, there must be different ways to execute the subquery depending on the value of `outer_expr`. In MySQL 5.1 before 5.1.16, the optimizer chose speed over distinguishing a `NULL` from `FALSE` result, so for some queries, you might get a `FALSE` result rather than `NULL`.

As of MySQL 5.1.16, the optimizer chooses SQL compliance over speed, so it accounts for the possibility that `outer_expr` might be `NULL`.

If `outer_expr` is `NULL`, to evaluate the following expression, it is necessary to run the `SELECT` to determine whether it produces any rows:

```
NULL IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

It is necessary to run the original `SELECT` here, without any pushed-down equalities of the kind mentioned earlier.

On the other hand, when `outer_expr` is not `NULL`, it is absolutely essential that this comparison:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

be converted to this expression that uses a pushed-down condition:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

Without this conversion, subqueries will be slow. To solve the dilemma of whether to push down or not push down conditions into the

subquery, the conditions are wrapped in “trigger” functions. Thus, an expression of the following form:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

is converted into:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
        AND trigcond(outer_expr=inner_expr))
```

More generally, if the subquery comparison is based on several pairs of outer and inner expressions, the conversion takes this comparison:

```
(oe_1, ..., oe_N) IN (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

and converts it to this expression:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
        AND trigcond(oe_1=ie_1)
        AND ...
        AND trigcond(oe_N=ie_N)
    )
```

Each `trigcond(X)` is a special function that evaluates to the following values:

- `X` when the “linked” outer expression `oe_i` is not `NULL`
- `TRUE` when the “linked” outer expression `oe_i` is `NULL`

Note that trigger functions are *not* triggers of the kind that you create with `CREATE TRIGGER`.

Equalities that are wrapped into `trigcond()` functions are not first class predicates for the query optimizer. Most optimizations cannot deal with predicates that may be turned on and off at query execution time, so they assume any `trigcond(X)` to be an unknown function and ignore it. At the moment, triggered equalities can be used by those optimizations:

- Reference optimizations: `trigcond(X=Y [OR Y IS NULL])` can be used to construct `ref`, `eq_ref`, or `ref_or_null` table accesses.
- Index lookup-based subquery execution engines: `trigcond(X=Y)` can be used to construct `unique_subquery` or `index_subquery` accesses.
- Table-condition generator: If the subquery is a join of several tables, the triggered condition will be checked as soon as possible.

When the optimizer uses a triggered condition to create some kind of index lookup-based access (as for the first two items of the preceding list), it must have a fallback strategy for the case when the condition is turned off. This fallback strategy is always the same: Do a full table scan. In `EXPLAIN` output, the fallback shows up as `Full scan on NULL key` in the `Extra` column:

```
mysql> EXPLAIN SELECT t1.col1,
-> t1.col1 IN (SELECT t2.key1 FROM t2 WHERE t2.col2=t1.col2) FROM t1\G
***** 1. row *****
      id: 1
    select_type: PRIMARY
      table: t1
      ...
***** 2. row *****
      id: 2
    select_type: DEPENDENT SUBQUERY
      table: t2
      type: index_subquery
possible_keys: key1
       key: key1
    key_len: 5
       ref: func
        rows: 2
      Extra: Using where; Full scan on NULL key
```

If you run `EXPLAIN EXTENDED` followed by `SHOW WARNINGS`, you can see the triggered condition:

```

***** 1. row *****
Level: Note
Code: 1003
Message: select `test`.`t1`.`col1` AS `col1`,
<in_optimizer>(`test`.`t1`.`col1`,
<exists>(<index_lookup>(<cache>(`test`.`t1`.`col1`) in t2
on key1 checking NULL
where (`test`.`t2`.`col2` = `test`.`t1`.`col2`) having
trigcond(<is_not_null_test>(`test`.`t2`.`key1`)))) AS
`t1.col1 IN (select t2.key1 from t2 where t2.col2=t1.col2)`
from `test`.`t1`

```

The use of triggered conditions has some performance implications. A `NULL IN (SELECT ...)` expression now may cause a full table scan (which is slow) when it previously did not. This is the price paid for correct results (the goal of the trigger-condition strategy was to improve compliance and not speed).

For multiple-table subqueries, execution of `NULL IN (SELECT ...)` will be particularly slow because the join optimizer doesn't optimize for the case where the outer expression is `NULL`. It assumes that subquery evaluations with `NULL` on the left side are very rare, even if there are statistics that indicate otherwise. On the other hand, if the outer expression might be `NULL` but never actually is, there is no performance penalty.

To help the query optimizer better execute your queries, use these tips:

- A column must be declared as `NOT NULL` if it really is. (This also helps other aspects of the optimizer.)
- If you don't need to distinguish a `NULL` from `FALSE` subquery result, you can easily avoid the slow execution path. Replace a comparison that looks like this:

```
outer_expr IN (SELECT inner_expr FROM ...)
```

with this expression:

```
(outer_expr IS NOT NULL) AND (outer_expr IN (SELECT inner_expr FROM ...))
```

Then `NULL IN (SELECT ...)` will never be evaluated because MySQL stops evaluating `AND` parts as soon as the expression result is clear.

6.2.15. LIMIT Optimization

In some cases, MySQL handles a query differently when you are using `LIMIT row_count` and not using `HAVING`:

- If you are selecting only a few rows with `LIMIT`, MySQL uses indexes in some cases when normally it would prefer to do a full table scan.
- If you use `LIMIT row_count` with `ORDER BY`, MySQL ends the sorting as soon as it has found the first `row_count` rows of the sorted result, rather than sorting the entire result. If ordering is done by using an index, this is very fast. If a filesort must be done, all rows that match the query without the `LIMIT` clause must be selected, and most or all of them must be sorted, before it can be ascertained that the first `row_count` rows have been found. In either case, after the initial rows have been found, there is no need to sort any remainder of the result set, and MySQL does not do so.
- When combining `LIMIT row_count` with `DISTINCT`, MySQL stops as soon as it finds `row_count` unique rows.
- In some cases, a `GROUP BY` can be resolved by reading the key in order (or doing a sort on the key) and then calculating summaries until the key value changes. In this case, `LIMIT row_count` does not calculate any unnecessary `GROUP BY` values.
- As soon as MySQL has sent the required number of rows to the client, it aborts the query unless you are using `SQL_CALC_FOUND_ROWS`.
- `LIMIT 0` quickly returns an empty set. This can be useful for checking the validity of a query. When using one of the MySQL APIs, it can also be employed for obtaining the types of the result columns. (This trick does not work in the MySQL Monitor (the `mysql` program), which merely displays `Empty set` in such cases; you should instead use `SHOW COLUMNS` or `DESCRIBE` for this purpose.)
- When the server uses temporary tables to resolve the query, it uses the `LIMIT row_count` clause to calculate how much space is

required.

6.2.16. How to Avoid Table Scans

The output from `EXPLAIN` shows `ALL` in the `type` column when MySQL uses a table scan to resolve a query. This usually happens under the following conditions:

- The table is so small that it is faster to perform a table scan than to bother with a key lookup. This is common for tables with fewer than 10 rows and a short row length.
- There are no usable restrictions in the `ON` or `WHERE` clause for indexed columns.
- You are comparing indexed columns with constant values and MySQL has calculated (based on the index tree) that the constants cover too large a part of the table and that a table scan would be faster. See [Section 6.2.4, “WHERE Clause Optimization”](#).
- You are using a key with low cardinality (many rows match the key value) through another column. In this case, MySQL assumes that by using the key it probably will do many key lookups and that a table scan would be faster.

MySQL Enterprise

For expert advice on avoiding excessive table scans subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

For small tables, a table scan often is appropriate and the performance impact is negligible. For large tables, try the following techniques to avoid having the optimizer incorrectly choose a table scan:

- Use `ANALYZE TABLE tbl_name` to update the key distributions for the scanned table. See [Section 12.5.2.1, “ANALYZE TABLE Syntax”](#).
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

See [Section 12.2.7.2, “Index Hint Syntax”](#).

- Start `mysqld` with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See [Section 5.2.3, “System Variables”](#).

6.2.17. Speed of INSERT Statements

The time required for inserting a row is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)
- Sending query to server: (2)
- Parsing query: (2)
- Inserting row: (1 × size of row)
- Inserting indexes: (1 × number of indexes)
- Closing: (1)

This does not take into consideration the initial overhead to open tables, which is done once for each concurrently running query.

The size of the table slows down the insertion of indexes by $\log N$, assuming B-tree indexes.

You can use the following methods to speed up inserts:

- If you are inserting many rows from the same client at the same time, use `INSERT` statements with multiple `VALUES` lists to insert several rows at a time. This is considerably faster (many times faster in some cases) than using separate single-row `INSERT` statements. If you are adding data to a non-empty table, you can tune the `bulk_insert_buffer_size` variable to make data insertion even faster. See [Section 5.2.3, “System Variables”](#).
- If multiple clients are inserting a lot of rows, you can get higher speed by using the `INSERT DELAYED` statement. See [Section 12.2.4.2, “INSERT DELAYED Syntax”](#).
- For a `MyISAM` table, you can use concurrent inserts to add rows at the same time that `SELECT` statements are running, if there are no deleted rows in middle of the data file. See [Section 6.3.3, “Concurrent Inserts”](#).
- When loading a table from a text file, use `LOAD DATA INFILE`. This is usually 20 times faster than using `INSERT` statements. See [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).
- With some extra work, it is possible to make `LOAD DATA INFILE` run even faster for a `MyISAM` table when the table has many indexes. Use the following procedure:
 1. Optionally create the table with `CREATE TABLE`.
 2. Execute a `FLUSH TABLES` statement or a `mysqladmin flush-tables` command.
 3. Use `myisamchk --keys-used=0 -rq /path/to/db/tbl_name`. This removes all use of indexes for the table.
 4. Insert data into the table with `LOAD DATA INFILE`. This does not update any indexes and therefore is very fast.
 5. If you intend only to read from the table in the future, use `myisampack` to compress it. See [Section 13.4.3.3, “Compressed Table Characteristics”](#).
 6. Re-create the indexes with `myisamchk -rq /path/to/db/tbl_name`. This creates the index tree in memory before writing it to disk, which is much faster than updating the index during `LOAD DATA INFILE` because it avoids lots of disk seeks. The resulting index tree is also perfectly balanced.
 7. Execute a `FLUSH TABLES` statement or a `mysqladmin flush-tables` command.

`LOAD DATA INFILE` performs the preceding optimization automatically if the `MyISAM` table into which you insert data is empty. The main difference between automatic optimization and using the procedure explicitly is that you can let `myisamchk` allocate much more temporary memory for the index creation than you might want the server to allocate for index re-creation when it executes the `LOAD DATA INFILE` statement.

You can also disable or enable the indexes for a `MyISAM` table by using the following statements rather than `myisamchk`. If you use these statements, you can skip the `FLUSH TABLE` operations:

```
ALTER TABLE tbl_name DISABLE KEYS;
ALTER TABLE tbl_name ENABLE KEYS;
```

- To speed up `INSERT` operations that are performed with multiple statements for non-transactional tables, lock your tables:

```
LOCK TABLES a WRITE;
INSERT INTO a VALUES (1,23),(2,34),(4,33);
INSERT INTO a VALUES (8,26),(6,29);
...
UNLOCK TABLES;
```

This benefits performance because the index buffer is flushed to disk only once, after all `INSERT` statements have completed. Normally, there would be as many index buffer flushes as there are `INSERT` statements. Explicit locking statements are not needed if you can insert all rows with a single `INSERT`.

To obtain faster insertions for transactional tables, you should use `START TRANSACTION` and `COMMIT` instead of `LOCK TABLES`.

Locking also lowers the total time for multiple-connection tests, although the maximum wait time for individual connections might go up because they wait for locks. Suppose that five clients attempt to perform inserts simultaneously as follows:

- Connection 1 does 1000 inserts
- Connections 2, 3, and 4 do 1 insert
- Connection 5 does 1000 inserts

If you do not use locking, connections 2, 3, and 4 finish before 1 and 5. If you use locking, connections 2, 3, and 4 probably do not finish before 1 or 5, but the total time should be about 40% faster.

`INSERT`, `UPDATE`, and `DELETE` operations are very fast in MySQL, but you can obtain better overall performance by adding locks around everything that does more than about five successive inserts or updates. If you do very many successive inserts, you could do a `LOCK TABLES` followed by an `UNLOCK TABLES` once in a while (each 1,000 rows or so) to allow other threads access to the table. This would still result in a nice performance gain.

`INSERT` is still much slower for loading data than `LOAD DATA INFILE`, even when using the strategies just outlined.

- To increase performance for `MyISAM` tables, for both `LOAD DATA INFILE` and `INSERT`, enlarge the key cache by increasing the `key_buffer_size` system variable. See [Section 6.5.2, “Tuning Server Parameters”](#).

MySQL Enterprise

For more advice on optimizing the performance of your server, subscribe to the MySQL Enterprise Monitor. Numerous advisors are dedicated to monitoring performance. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

6.2.18. Speed of `UPDATE` Statements

An update statement is optimized like a `SELECT` query with the additional overhead of a write. The speed of the write depends on the amount of data being updated and the number of indexes that are updated. Indexes that are not changed do not get updated.

Another way to get fast updates is to delay updates and then do many updates in a row later. Performing multiple updates together is much quicker than doing one at a time if you lock the table.

For a `MyISAM` table that uses dynamic row format, updating a row to a longer total length may split the row. If you do this often, it is very important to use `OPTIMIZE TABLE` occasionally. See [Section 12.5.2.5, “OPTIMIZE TABLE Syntax”](#).

6.2.19. Speed of `DELETE` Statements

The time required to delete individual rows is exactly proportional to the number of indexes. To delete rows more quickly, you can increase the size of the key cache by increasing the `key_buffer_size` system variable. See [Section 6.5.2, “Tuning Server Parameters”](#).

To delete all rows from a table, `TRUNCATE TABLE tbl_name` is faster than `DELETE FROM tbl_name`. Truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction or active table lock. See [Section 12.2.9, “TRUNCATE Syntax”](#).

6.2.20. Other Optimization Tips

This section lists a number of miscellaneous tips for improving query processing speed:

- Use persistent connections to the database to avoid connection overhead. If you cannot use persistent connections and you are initiating many new connections to the database, you may want to change the value of the `thread_cache_size` variable. See [Section 6.5.2, “Tuning Server Parameters”](#).
- Always check whether all your queries really use the indexes that you have created in the tables. In MySQL, you can do this with the `EXPLAIN` statement. See [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#).
- Try to avoid complex `SELECT` queries on `MyISAM` tables that are updated frequently, to avoid problems with table locking that occur due to contention between readers and writers.
- `MyISAM` supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. If it is important to be able to do this, you should consider using the table

in ways that avoid deleting rows. Another possibility is to run `OPTIMIZE TABLE` to defragment the table after you have deleted a lot of rows from it. This behavior is altered by setting the `concurrent_insert` variable. You can force new rows to be appended (and therefore allow concurrent inserts), even in tables that have deleted rows. See [Section 6.3.3, “Concurrent Inserts”](#).

MySQL Enterprise

For optimization tips geared to your specific circumstances subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- To fix any compression issues that may have occurred with `ARCHIVE` tables, you can use `OPTIMIZE TABLE`. See [Section 13.10, “The ARCHIVE Storage Engine”](#).
- Use `ALTER TABLE ... ORDER BY expr1, expr2, ...` if you usually retrieve rows in `expr1, expr2, ...` order. By using this option after extensive changes to the table, you may be able to get higher performance.
- In some cases, it may make sense to introduce a column that is “hashed” based on information from other columns. If this column is short, reasonably unique, and indexed, it may be much faster than a “wide” index on many columns. In MySQL, it is very easy to use this extra column:

```
SELECT * FROM tbl_name
WHERE hash_col=MD5(CONCAT(col1,col2))
AND col1='constant' AND col2='constant';
```

- For `MyISAM` tables that change frequently, you should try to avoid all variable-length columns (`VARCHAR`, `BLOB`, and `TEXT`). The table uses dynamic row format if it includes even a single variable-length column. See [Chapter 13, Storage Engines](#).
- It is normally not useful to split a table into different tables just because the rows become large. In accessing a row, the biggest performance hit is the disk seek needed to find the first byte of the row. After finding the data, most modern disks can read the entire row fast enough for most applications. The only cases where splitting up a table makes an appreciable difference is if it is a `MyISAM` table using dynamic row format that you can change to a fixed row size, or if you very often need to scan the table but do not need most of the columns. See [Chapter 13, Storage Engines](#).
- If you often need to calculate results such as counts based on information from a lot of rows, it may be preferable to introduce a new table and update the counter in real time. An update of the following form is very fast:

```
UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
```

This is very important when you use MySQL storage engines such as `MyISAM` that has only table-level locking (multiple readers with single writers). This also gives better performance with most database systems, because the row locking manager in this case has less to do.

- If you need to collect statistics from large log tables, use summary tables instead of scanning the entire log table. Maintaining the summaries should be much faster than trying to calculate statistics “live.” Regenerating new summary tables from the logs when things change (depending on business decisions) is faster than changing the running application.
- If possible, you should classify reports as “live” or as “statistical,” where data needed for statistical reports is created only from summary tables that are generated periodically from the live data.
- Take advantage of the fact that columns have default values. Insert values explicitly only when the value to be inserted differs from the default. This reduces the parsing that MySQL must do and improves the insert speed.
- In some cases, it is convenient to pack and store data into a `BLOB` column. In this case, you must provide code in your application to pack and unpack information, but this may save a lot of accesses at some stage. This is practical when you have data that does not conform well to a rows-and-columns table structure.
- Normally, you should try to keep all data non-redundant (observing what is referred to in database theory as *third normal form*). However, there may be situations in which it can be advantageous to duplicate information or create summary tables to gain more speed.
- Stored routines or UDFs (user-defined functions) may be a good way to gain performance for some tasks. See [Chapter 19, Stored Procedures and Functions](#), and [Section 28.3, “Adding New Functions to MySQL”](#), for more information.
- You can increase performance by caching queries or answers in your application and then executing many inserts or updates together. If your database system supports table locks (as do MySQL and Oracle), this should help to ensure that the index cache is only flushed once after all updates. You can also take advantage of MySQL’s query cache to achieve similar results; see [Section 6.5.4, “The MySQL Query Cache”](#).

- Use `INSERT DELAYED` when you do not need to know when your data is written. This reduces the overall insertion impact because many rows can be written with a single disk write.
- Use `INSERT LOW_PRIORITY` when you want to give `SELECT` statements higher priority than your inserts.

Use `SELECT HIGH_PRIORITY` to get retrievals that jump the queue. That is, the `SELECT` is executed even if there is another client waiting to do a write.

`LOW_PRIORITY` and `HIGH_PRIORITY` have an effect only for storage engines that use only table-level locking (`MyISAM`, `MEMORY`, `MERGE`).

- Use multiple-row `INSERT` statements to store many rows with one SQL statement. Many SQL servers support this, including MySQL.
- Use `LOAD DATA INFILE` to load large amounts of data. This is faster than using `INSERT` statements.
- Use `AUTO_INCREMENT` columns so that each row in a table can be identified by a single unique value.
- Use `OPTIMIZE TABLE` once in a while to avoid fragmentation with dynamic-format `MyISAM` tables. See [Section 13.4.3, “MyISAM Table Storage Formats”](#).
- Use `MEMORY` tables when possible to get more speed. See [Section 13.7, “The MEMORY \(HEAP\) Storage Engine”](#). `MEMORY` tables are useful for non-critical data that is accessed often, such as information about the last displayed banner for users who don't have cookies enabled in their Web browser. User sessions are another alternative available in many Web application environments for handling volatile state data.
- With Web servers, images and other binary assets should normally be stored as files. That is, store only a reference to the file rather than the file itself in the database. Most Web servers are better at caching files than database contents, so using files is generally faster.
- Columns with identical information in different tables should be declared to have identical data types so that joins based on the corresponding columns will be faster.
- Try to keep column names simple. For example, in a table named `customer`, use a column name of `name` instead of `customer_name`. To make your names portable to other SQL servers, you should keep them shorter than 18 characters.
- If you need really high speed, you should take a look at the low-level interfaces for data storage that the different SQL servers support. For example, by accessing the MySQL `MyISAM` storage engine directly, you could get a speed increase of two to five times compared to using the SQL interface. To be able to do this, the data must be on the same server as the application, and usually it should only be accessed by one process (because external file locking is really slow). One could eliminate these problems by introducing low-level `MyISAM` commands in the MySQL server (this could be one easy way to get more performance if needed). By carefully designing the database interface, it should be quite easy to support this type of optimization.
- If you are using numerical data, it is faster in many cases to access information from a database (using a live connection) than to access a text file. Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses. You also save code in your application because you need not parse your text files to find line and column boundaries.
- Replication can provide a performance benefit for some operations. You can distribute client retrievals among replication servers to split up the load. To avoid slowing down the master while making backups, you can make backups using a slave server. See [Chapter 15, Replication](#).
- Declaring a `MyISAM` table with the `DELAY_KEY_WRITE=1` table option makes index updates faster because they are not flushed to disk until the table is closed. The downside is that if something kills the server while such a table is open, you should ensure that the table is okay by running the server with the `--myisam-recover` option, or by running `myisamchk` before restarting the server. (However, even in this case, you should not lose anything by using `DELAY_KEY_WRITE`, because the key information can always be generated from the data rows.)

6.3. Locking Issues

MySQL manages contention for table contents using locking:

- Internal locking is performed within the MySQL server itself to manage contention for table contents by multiple threads. This type

of locking is internal because it is performed entirely by the server and involves no other programs. See [Section 6.3.1, “Internal Locking Methods”](#).

- External locking occurs when the server and other programs lock table files to coordinate among themselves which program can access the tables at which time. See [Section 6.3.4, “External Locking”](#). See [Section 6.3.4, “External Locking”](#).

6.3.1. Internal Locking Methods

This section discusses internal locking; that is, locking performed within the MySQL server itself to manage contention for table contents by multiple threads. This type of locking is internal because it is performed entirely by the server and involves no other programs. External locking occurs when the server and other programs lock table files to coordinate among themselves which program can access the tables at which time. See [Section 6.3.4, “External Locking”](#).

MySQL uses table-level locking for **MyISAM** and **MEMORY** tables, and row-level locking for **InnoDB** tables.

In many cases, you can make an educated guess about which locking type is best for an application, but generally it is difficult to say that a given lock type is better than another. Everything depends on the application and different parts of an application may require different lock types.

To decide whether you want to use a storage engine with row-level locking, you should look at what your application does and what mix of select and update statements it uses. For example, most Web applications perform many selects, relatively few deletes, updates based mainly on key values, and inserts into a few specific tables. The base MySQL **MyISAM** setup is very well tuned for this.

MySQL Enterprise

The MySQL Enterprise Monitor provides expert advice on when to use table-level locking and when to use row-level locking. To subscribe see <http://www.mysql.com/products/enterprise/advisors.html>.

Table locking in MySQL is deadlock-free for storage engines that use table-level locking. Deadlock avoidance is managed by always requesting all needed locks at once at the beginning of a query and always locking the tables in the same order.

MySQL grants table **WRITE** locks as follows:

1. If there are no locks on the table, put a write lock on it.
2. Otherwise, put the lock request in the write lock queue.

MySQL grants table **READ** locks as follows:

1. If there are no write locks on the table, put a read lock on it.
2. Otherwise, put the lock request in the read lock queue.

When a lock is released, the lock is made available to the threads in the write lock queue and then to the threads in the read lock queue. This means that if you have many updates for a table, **SELECT** statements wait until there are no more updates.

You can analyze the table lock contention on your system by checking the **Table_locks_waited** and **Table_locks_immediate** status variables:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Table_locks_immediate | 1151552 |
| Table_locks_waited    | 15324  |
+-----+-----+
```

The **MyISAM** storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a **MyISAM** table has no free blocks in the middle of the data file, rows are always inserted at the end of the data file. In this case, you can freely mix concurrent **INSERT** and **SELECT** statements for a **MyISAM** table without locks. That is, you can insert rows into a **MyISAM** table at the same time other clients are reading from it. Holes can result from rows having been deleted from or updated in the middle of the table. If there are holes, concurrent inserts are disabled but are re-enabled automatically when all holes have been filled with new data.. This behavior is altered by the **concurrent_insert** system variable. See [Section 6.3.3, “Concurrent Inserts”](#).

If you want to perform many `INSERT` and `SELECT` operations on a table `real_table` when concurrent inserts are not possible, you can insert rows into a temporary table `temp_table` and update the real table with the rows from the temporary table periodically. This can be done with the following code:

```
mysql> LOCK TABLES real_table WRITE, temp_table WRITE;
mysql> INSERT INTO real_table SELECT * FROM temp_table;
mysql> DELETE FROM temp_table;
mysql> UNLOCK TABLES;
```

InnoDB uses row locks. Deadlocks are possible for **InnoDB** because it automatically acquires locks during the processing of SQL statements, not at the start of the transaction.

Advantages of row-level locking:

- Fewer lock conflicts when accessing different rows in many threads
- Fewer changes for rollbacks
- Possible to lock a single row for a long time

Disadvantages of row-level locking:

- Requires more memory than table-level locks
- Slower than table-level locks when used on a large part of the table because you must acquire many more locks
- Definitely much slower than other locks if you often do `GROUP BY` operations on a large part of the data or if you must scan the entire table frequently

Table locks are superior to row-level locks in the following cases:

- Most statements for the table are reads
- Statements for the table are a mix of reads and writes, where writes are updates or deletes for a single row that can be fetched with one key read:

```
UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
DELETE FROM tbl_name WHERE unique_key_col=key_value;
```

- `SELECT` combined with concurrent `INSERT` statements, and very few `UPDATE` or `DELETE` statements
- Many scans or `GROUP BY` operations on the entire table without any writers

With higher-level locks, you can more easily tune applications by supporting locks of different types, because the lock overhead is less than for row-level locks.

Options other than row-level locking:

- Versioning (such as that used in MySQL for concurrent inserts) where it is possible to have one writer at the same time as many readers. This means that the database or table supports different views for the data depending on when access begins. Other common terms for this are “time travel,” “copy on write,” or “copy on demand.”
- Copy on demand is in many cases superior to row-level locking. However, in the worst case, it can use much more memory than using normal locks.
- Instead of using row-level locks, you can employ application-level locks, such as those provided by `GET_LOCK()` and `RELEASE_LOCK()` in MySQL. These are advisory locks, so they work only in well-behaved applications. See [Section 11.11.4, “Miscellaneous Functions”](#).

6.3.2. Table Locking Issues

To achieve a very high lock speed, MySQL uses table locking (instead of page, row, or column locking) for all storage engines except [InnoDB](#) and [NDBCLUSTER](#).

For [InnoDB](#) tables, MySQL only uses table locking if you explicitly lock the table with [LOCK TABLES](#). For these storage engines, we recommend that you not use [LOCK TABLES](#) at all, because [InnoDB](#) uses automatic row-level locking to ensure transaction isolation.

For large tables, table locking is much better than row locking for most applications, but there are some pitfalls:

- Table locking enables many threads to read from a table at the same time, but if a thread wants to write to a table, it must first get exclusive access. During the update, all other threads that want to access this particular table must wait until the update is done.
- Table updates normally are considered to be more important than table retrievals, so they are given higher priority. This should ensure that updates to a table are not “starved” even if there is heavy [SELECT](#) activity for the table.
- Table locking causes problems in cases such as when a thread is waiting because the disk is full and free space needs to become available before the thread can proceed. In this case, all threads that want to access the problem table are also put in a waiting state until more disk space is made available.

Table locking is also disadvantageous under the following scenario:

- A client issues a [SELECT](#) that takes a long time to run.
- Another client then issues an [UPDATE](#) on the same table. This client waits until the [SELECT](#) is finished.
- Another client issues another [SELECT](#) statement on the same table. Because [UPDATE](#) has higher priority than [SELECT](#), this [SELECT](#) waits for the [UPDATE](#) to finish, *and* for the first [SELECT](#) to finish.

The following items describe some ways to avoid or reduce contention caused by table locking:

- Try to get the [SELECT](#) statements to run faster so that they lock tables for a shorter time. You might have to create some summary tables to do this.
- Start `mysqld` with `--low-priority-updates`. For storage engines that use only table-level locking ([MyISAM](#), [MEMORY](#), [MERGE](#)), this gives all statements that update (modify) a table lower priority than [SELECT](#) statements. In this case, the second [SELECT](#) statement in the preceding scenario would execute before the [UPDATE](#) statement, and would not need to wait for the first [SELECT](#) to finish.
- You can specify that all updates issued in a specific connection should be done with low priority by using the `SET LOW_PRIORITY_UPDATES=1` statement. See [Section 12.5.3, “SET Syntax”](#).
- You can give a specific [INSERT](#), [UPDATE](#), or [DELETE](#) statement lower priority with the `LOW_PRIORITY` attribute.
- You can give a specific [SELECT](#) statement higher priority with the `HIGH_PRIORITY` attribute. See [Section 12.2.7, “SELECT Syntax”](#).
- You can start `mysqld` with a low value for the `max_write_lock_count` system variable to force MySQL to temporarily elevate the priority of all [SELECT](#) statements that are waiting for a table after a specific number of inserts to the table occur. This allows [READ](#) locks after a certain number of [WRITE](#) locks.
- If you have problems with [INSERT](#) combined with [SELECT](#), you might want to consider switching to [MyISAM](#) tables, which support concurrent [SELECT](#) and [INSERT](#) statements. (See [Section 6.3.3, “Concurrent Inserts”](#).)
- If you mix inserts and deletes on the same table, `INSERT DELAYED` may be of great help. See [Section 12.2.4.2, “INSERT DELAYED Syntax”](#).
- If you have problems with mixed [SELECT](#) and [DELETE](#) statements, the `LIMIT` option to [DELETE](#) may help. See [Section 12.2.1, “DELETE Syntax”](#).
- Using `SQL_BUFFER_RESULT` with [SELECT](#) statements can help to make the duration of table locks shorter. See [Section 12.2.7, “SELECT Syntax”](#).

- You could change the locking code in `mysys/thr_lock.c` to use a single queue. In this case, write locks and read locks would have the same priority, which might help some applications.

Here are some tips concerning table locks in MySQL:

- Concurrent users are not a problem if you do not mix updates with selects that need to examine many rows in the same table.
- You can use `LOCK TABLES` to increase speed, because many updates within a single lock is much faster than updating without locks. Splitting table contents into separate tables may also help.
- If you encounter speed problems with table locks in MySQL, you may be able to improve performance by converting some of your tables to `InnoDB`. See [Section 13.5, “The InnoDB Storage Engine”](#).

MySQL Enterprise

Lock contention can seriously degrade performance. The MySQL Enterprise Monitor provides expert advice on avoiding this problem. To subscribe see <http://www.mysql.com/products/enterprise/advisors.html>.

6.3.3. Concurrent Inserts

The `MyISAM` storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a `MyISAM` table has no holes in the data file (deleted rows in the middle), inserts can be performed to add rows to the end of the table at the same time that `SELECT` statements are reading rows from the table.

The `concurrent_insert` system variable can be set to modify the concurrent-insert processing. By default, the variable is set to 1 and concurrent inserts are handled as just described. If `concurrent_inserts` is set to 0, concurrent inserts are disabled. If the variable is set to 2, concurrent inserts at the end of the table are allowed even for tables that have deleted rows. See also the description of the `concurrent_insert` system variable.

Under circumstances where concurrent inserts can be used, there is seldom any need to use the `DELAYED` modifier for `INSERT` statements. See [Section 12.2.4.2, “INSERT DELAYED Syntax”](#).

If you are using the binary log, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. See [Section 5.11.4, “The Binary Log”](#).

With `LOAD DATA INFILE`, if you specify `CONCURRENT` with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other threads can retrieve data from the table while `LOAD DATA` is executing. Use of the `CONCURRENT` option affects the performance of `LOAD DATA` a bit, even if no other thread is using the table at the same time.

If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` option if the server was started with that option. It also causes concurrent inserts not to be used.

For `LOCK TABLE`, the difference between `READ LOCAL` and `READ` is that `READ LOCAL` allows non-conflicting `INSERT` statements (concurrent inserts) to execute while the lock is held. However, this cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock.

6.3.4. External Locking

External locking is the use of filesystem locking to manage contention for database tables by multiple processes. External locking is used in situations where a single process such as the MySQL server cannot be assumed to be the only process that requires access to tables. Here are some examples:

- If you run multiple servers that use the same database directory (not recommended), each server must have external locking enabled.
- If you use `myisamchk` to perform table maintenance operations on `MyISAM` tables, you must either ensure that the server is not running, or that the server has external locking enabled so that it locks table files as necessary to coordinate with `myisamchk` for access to the tables. The same is true for use of `myisampack` to pack `MyISAM` tables.

With external locking in effect, each process that requires access to a table acquires a filesystem lock for the table files before proceeding to access the table. If all necessary locks cannot be acquired, the process is blocked from accessing the table until the locks can be

obtained (after the process that currently holds the locks releases them).

External locking affects server performance because the server must sometimes wait for other processes before it can access tables.

External locking is unnecessary if you run a single server to access a given data directory (which is the usual case) and if no other programs such as `myisamchk` need to modify tables while the server is running. If you only *read* tables with other programs, external locking is not required, although `myisamchk` might report warnings if the server changes tables while `myisamchk` is reading them.

With external locking disabled, to use `myisamchk`, you must either stop the server while `myisamchk` executes or else lock and flush the tables before running `myisamchk`. (See [Section 6.5.1, “System Factors and Startup Parameter Tuning”](#).) To avoid this requirement, use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables.

For `mysqld`, external locking is controlled by the value of the `skip_external_locking` system variable. (Before MySQL 4.0.3, this variable is named `skip_locking`.) When this variable is enabled, external locking is disabled, and vice versa. From MySQL 4.0 on, external locking is disabled by default. Before MySQL 4.0, external locking is enabled by default on Linux or when MySQL is configured to use MIT-pthreads.

Use of external locking can be controlled at server startup by using the `--external-locking` or `--skip-external-locking` option. (Before MySQL 4.0.3, these options are named `--enable-locking` and `--skip-locking`.)

If you do use external locking option to enable updates to `MyISAM` tables from many MySQL processes, you must ensure that the following conditions are satisfied:

- You should not use the query cache for queries that use tables that are updated by another process.
- You should not start the server with the `--delay-key-write=ALL` option or use the `DELAY_KEY_WRITE=1` table option for any shared tables. Otherwise, index corruption can occur.

The easiest way to satisfy these conditions is to always use `--external-locking` together with `--delay-key-write=OFF` and `--query-cache-size=0`. (This is not done by default because in many setups it is useful to have a mixture of the preceding options.)

6.4. Optimizing Database Structure

6.4.1. Design Choices

MySQL keeps row data and index data in separate files. Many (almost all) other database systems mix row and index data in the same file. We believe that the MySQL choice is better for a very wide range of modern systems.

Another way to store the row data is to keep the information for each column in a separate area (examples are SDBM and Focus). This causes a performance hit for every query that accesses more than one column. Because this degenerates so quickly when more than one column is accessed, we believe that this model is not good for general-purpose databases.

The more common case is that the index and data are stored together (as in Oracle/Sybase, et al). In this case, you find the row information at the leaf page of the index. The good thing with this layout is that it, in many cases, depending on how well the index is cached, saves a disk read. The bad things with this layout are:

- Table scanning is much slower because you have to read through the indexes to get at the data.
- You cannot use only the index table to retrieve data for a query.
- You use more space because you must duplicate indexes from the nodes (you cannot store the row in the nodes).
- Deletes degenerate the table over time (because indexes in nodes are usually not updated on delete).
- It is more difficult to cache only the index data.

6.4.2. Make Your Data as Small as Possible

One of the most basic optimizations is to design your tables to take as little space on the disk as possible. This can result in huge im-

provements because disk reads are faster, and smaller tables normally require less main memory while their contents are being actively processed during query execution. Indexing also is a lesser resource burden if done on smaller columns.

MySQL supports many different storage engines (table types) and row formats. For each table, you can decide which storage and indexing method to use. Choosing the proper table format for your application may give you a big performance gain. See [Chapter 13, Storage Engines](#).

You can get better performance for a table and minimize storage space by using the techniques listed here:

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory. For example, use the smaller integer types if possible to get smaller tables. `MEDIUMINT` is often a better choice than `INT` because a `MEDIUMINT` column uses 25% less space.
- Declare columns to be **NOT NULL** if possible. It makes everything faster and you save one bit per column. If you really need `NULL` in your application, you should definitely use it. Just avoid having it on all columns by default.
- For **MyISAM** tables, if you do not have any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB` columns), a fixed-size row format is used. This is faster but unfortunately may waste some space. See [Section 13.4.3, “MyISAM Table Storage Formats”](#). You can hint that you want to have fixed length rows even if you have `VARCHAR` columns with the `CREATE TABLE` option `ROW_FORMAT=FIXED`.
- **InnoDB** tables use a compact storage format. In versions of MySQL earlier than 5.0.3, **InnoDB** rows contain some redundant information, such as the number of columns and the length of each column, even for fixed-size columns. By default, tables are created in the compact format (`ROW_FORMAT=COMPACT`). If you wish to downgrade to older versions of MySQL, you can request the old format with `ROW_FORMAT=REDUNDANT`.

The presence of the compact row format decreases row storage space by about 20% at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed it is likely to be faster. If it is a rare case that is limited by CPU speed, it might be slower.

The compact **InnoDB** format also changes how `CHAR` columns containing UTF-8 data are stored. With `ROW_FORMAT=REDUNDANT`, a UTF-8 `CHAR(N)` occupies $3 \times N$ bytes, given that the maximum length of a UTF-8 encoded character is three bytes. Many languages can be written primarily using single-byte UTF-8 characters, so a fixed storage length often wastes space. With `ROW_FORMAT=COMPACT` format, **InnoDB** allocates a variable amount of storage in the range from N to $3 \times N$ bytes for these columns by stripping trailing spaces if necessary. The minimum storage length is kept as N bytes to facilitate in-place updates in typical cases.

- The primary index of a table should be as short as possible. This makes identification of each row easy and efficient.
- Create only the indexes that you really need. Indexes are good for retrieval but bad when you need to store data quickly. If you access a table mostly by searching on a combination of columns, create an index on them. The first part of the index should be the column most used. If you *always* use many columns when selecting from the table, you should use the column with more duplicates first to obtain better compression of the index.
- If it is very likely that a string column has a unique prefix on the first number of characters, it's better to index only this prefix, using MySQL's support for creating an index on the leftmost part of the column (see [Section 12.1.7, “CREATE INDEX Syntax”](#)). Shorter indexes are faster, not only because they require less disk space, but because they also give you more hits in the index cache, and thus fewer disk seeks. See [Section 6.5.2, “Tuning Server Parameters”](#).
- In some circumstances, it can be beneficial to split into two a table that is scanned very often. This is especially true if it is a dynamic-format table and it is possible to use a smaller static format table that can be used to find the relevant rows when scanning the table.

6.4.3. Column Indexes

All MySQL data types can be indexed. Use of indexes on the relevant columns is the best way to improve the performance of `SELECT` operations.

The maximum number of indexes per table and the maximum index length is defined per storage engine. See [Chapter 13, Storage Engines](#). All storage engines support at least 16 indexes per table and a total index length of at least 256 bytes. Most storage engines have higher limits.

With `col_name(N)` syntax in an index specification, you can create an index that uses only the first N characters of a string column.

Indexing only a prefix of column values in this way can make the index file much smaller. When you index a **BLOB** or **TEXT** column, you *must* specify a prefix length for the index. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for **InnoDB** tables). Note that prefix limits are measured in bytes, whereas the prefix length in **CREATE TABLE** statements is interpreted as number of characters. *Be sure to take this into account when specifying a prefix length for a column that uses a multi-byte character set.*

You can also create **FULLTEXT** indexes. These are used for full-text searches. Only the **MyISAM** storage engine supports **FULLTEXT** indexes and only for **CHAR**, **VARCHAR**, and **TEXT** columns. Indexing always takes place over the entire column and column prefix indexing is not supported. For details, see [Section 11.8, “Full-Text Search Functions”](#).

You can also create indexes on spatial data types. Currently, only **MyISAM** supports R-tree indexes on spatial types. Other storage engines use B-trees for indexing spatial types (except for **ARCHIVE** and **NDBCLUSTER**, which do not support spatial type indexing).

The **MEMORY** storage engine uses **HASH** indexes by default, but also supports **BTREE** indexes.

6.4.4. Multiple-Column Indexes

MySQL can create composite indexes (that is, indexes on multiple columns). An index may consist of up to 15 columns. For certain data types, you can index a prefix of the column (see [Section 6.4.3, “Column Indexes”](#)).

A multiple-column index can be considered a sorted array containing values that are created by concatenating the values of the indexed columns.

MySQL uses multiple-column indexes in such a way that queries are fast when you specify a known quantity for the first column of the index in a **WHERE** clause, even if you do not specify values for the other columns.

Suppose that a table has the following specification:

```
CREATE TABLE test (
  id          INT NOT NULL,
  last_name   CHAR(30) NOT NULL,
  first_name  CHAR(30) NOT NULL,
  PRIMARY KEY (id),
  INDEX name (last_name,first_name)
);
```

The **name** index is an index over the **last_name** and **first_name** columns. The index can be used for queries that specify values in a known range for **last_name**, or for both **last_name** and **first_name**. Therefore, the **name** index is used in the following queries:

```
SELECT * FROM test WHERE last_name='Widenius';
SELECT * FROM test
  WHERE last_name='Widenius' AND first_name='Michael';
SELECT * FROM test
  WHERE last_name='Widenius'
     AND (first_name='Michael' OR first_name='Monty');
SELECT * FROM test
  WHERE last_name='Widenius'
     AND first_name >='M' AND first_name < 'N';
```

However, the **name** index is *not* used in the following queries:

```
SELECT * FROM test WHERE first_name='Michael';
SELECT * FROM test
  WHERE last_name='Widenius' OR first_name='Michael';
```

The manner in which MySQL uses indexes to improve query performance is discussed further in [Section 6.4.5, “How MySQL Uses Indexes”](#).

6.4.5. How MySQL Uses Indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then

read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. If a table has 1,000 rows, this is at least 100 times faster than reading sequentially. If you need to access most of the rows, it is faster to read sequentially, because this minimizes disk seeks.

Most MySQL indexes ([PRIMARY KEY](#), [UNIQUE](#), [INDEX](#), and [FULLTEXT](#)) are stored in B-trees. Exceptions are that indexes on spatial data types use R-trees, and that [MEMORY](#) tables also support hash indexes.

Strings are automatically prefix- and end-space compressed. See [Section 12.1.7, “CREATE INDEX Syntax”](#).

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in [MEMORY](#) tables) are described at the end of this section.

MySQL uses indexes for these operations:

- To find the rows matching a [WHERE](#) clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows.
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, [VARCHAR](#) and [CHAR](#) are considered the same if they are declared as the same size. For example, [VARCHAR\(10\)](#) and [CHAR\(10\)](#) are the same size, but [VARCHAR\(10\)](#) and [CHAR\(15\)](#) are not.

Comparison of dissimilar columns may prevent use of indexes if values cannot be compared directly without conversion. Suppose that a numeric column is compared to a string column. For a given value such as [1](#) in the numeric column, it might compare equal to any number of values in the string column such as ['1'](#), [' 1'](#), ['00001'](#), or ['01.e1'](#). This rules out use of any indexes for the string column.

- To find the [MIN\(\)](#) or [MAX\(\)](#) value for a specific indexed column [key_col](#). This is optimized by a preprocessor that checks whether you are using [WHERE key_part_N = constant](#) on all key parts that occur before [key_col](#) in the index. In this case, MySQL does a single key lookup for each [MIN\(\)](#) or [MAX\(\)](#) expression and replaces it with a constant. If all expressions are replaced with constants, the query returns at once. For example:

```
SELECT MIN(key\_part2), MAX(key\_part2)
FROM tbl\_name WHERE key\_part1=10;
```

- To sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable key (for example, [ORDER BY key_part1, key_part2](#)). If all key parts are followed by [DESC](#), the key is read in reverse order. See [Section 6.2.11, “ORDER BY Optimization”](#).
- In some cases, a query can be optimized to retrieve values without consulting the data rows. If a query uses only columns from a table that are numeric and that form a leftmost prefix for some key, the selected values may be retrieved from the index tree for greater speed:

```
SELECT key\_part3 FROM tbl\_name
WHERE key\_part1=1
```

Suppose that you issue the following [SELECT](#) statement:

```
mysql> SELECT * FROM tbl\_name WHERE col1=val1 AND col2=val2;
```

If a multiple-column index exists on [col1](#) and [col2](#), the appropriate rows can be fetched directly. If separate single-column indexes exist on [col1](#) and [col2](#), the optimizer tries to find the most restrictive index by deciding which index finds fewer rows and using that index to fetch the rows.

If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to find rows. For example, if you have a three-column index on ([col1](#), [col2](#), [col3](#)), you have indexed search capabilities on ([col1](#)), ([col1](#), [col2](#)), and ([col1](#), [col2](#), [col3](#)).

MySQL cannot use an index if the columns do not form a leftmost prefix of the index. Suppose that you have the [SELECT](#) statements shown here:

```

SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;

```

If an index exists on (col1, col2, col3), only the first two queries use the index. The third and fourth queries do involve indexed columns, but (col2) and (col2, col3) are not leftmost prefixes of (col1, col2, col3).

A B-tree index can be used for column comparisons in expressions that use the =, >, >=, <, <=, or BETWEEN operators. The index also can be used for LIKE comparisons if the argument to LIKE is a constant string that does not start with a wildcard character. For example, the following SELECT statements use indexes:

```

SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%ck%';

```

In the first statement, only rows with 'Patrick' <= key_col < 'Patricl' are considered. In the second statement, only rows with 'Pat' <= key_col < 'Pau' are considered.

The following SELECT statements do not use indexes:

```

SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;

```

In the first statement, the LIKE value begins with a wildcard character. In the second statement, the LIKE value is not a constant.

If you use ... LIKE '%string%' and string is longer than three characters, MySQL uses the *Turbo Boyer-Moore algorithm* to initialize the pattern for the string and then uses this pattern to perform the search more quickly.

A search using col_name IS NULL employs indexes if col_name is indexed.

Any index that does not span all AND levels in the WHERE clause is not used to optimize the query. In other words, to be able to use an index, a prefix of the index must be used in every AND group.

The following WHERE clauses use indexes:

```

... WHERE index_part1=1 AND index_part2=2 AND other_column=3
/* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2
/* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5
/* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;

```

These WHERE clauses do not use indexes:

```

/* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2

/* Index is not used in both parts of the WHERE clause */
... WHERE index=1 OR A=10

/* No index spans all rows */
... WHERE index_part1=1 OR index_part2=10

```

Sometimes MySQL does not use an index, even if one is available. One circumstance under which this occurs is when the optimizer estimates that using the index would require MySQL to access a very large percentage of the rows in the table. (In this case, a table scan is likely to be much faster because it requires fewer seeks.) However, if such a query uses LIMIT to retrieve only some of the rows, MySQL uses an index anyway, because it can much more quickly find the few rows to return in the result.

Hash indexes have somewhat different characteristics from those just discussed:

- They are used only for equality comparisons that use the = or <=> operators (but are very fast). They are not used for comparison operators such as < that find a range of values.
- The optimizer cannot use a hash index to speed up ORDER BY operations. (This type of index cannot be used to search for the next entry in order.)

- MySQL cannot determine approximately how many rows there are between two values (this is used by the range optimizer to decide which index to use). This may affect some queries if you change a **MyISAM** table to a hash-indexed **MEMORY** table.
- Only whole keys can be used to search for a row. (With a B-tree index, any leftmost prefix of the key can be used to find rows.)

MySQL Enterprise

Often, it is not possible to predict exactly what indexes will be required or will be most efficient — actual table usage is the best indicator. The MySQL Enterprise Monitor provides expert advice on this topic. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

6.4.6. The **MyISAM** Key Cache

To minimize disk I/O, the **MyISAM** storage engine exploits a strategy that is used by many database management systems. It employs a cache mechanism to keep the most frequently accessed table blocks in memory:

- For index blocks, a special structure called the *key cache* (or *key buffer*) is maintained. The structure contains a number of block buffers where the most-used index blocks are placed.
- For data blocks, MySQL uses no special cache. Instead it relies on the native operating system filesystem cache.

This section first describes the basic operation of the **MyISAM** key cache. Then it discusses features that improve key cache performance and that enable you to better control cache operation:

- Multiple threads can access the cache concurrently.
- You can set up multiple key caches and assign table indexes to specific caches.

To control the size of the key cache, use the `key_buffer_size` system variable. If this variable is set equal to zero, no key cache is used. The key cache also is not used if the `key_buffer_size` value is too small to allocate the minimal number of block buffers (8).

MySQL Enterprise

For expert advice on identifying the optimum size for `key_buffer_size`, subscribe to the MySQL Enterprise Monitor. See <http://www.mysql.com/products/enterprise/advisors.html>.

When the key cache is not operational, index files are accessed using only the native filesystem buffering provided by the operating system. (In other words, table index blocks are accessed using the same strategy as that employed for table data blocks.)

An index block is a contiguous unit of access to the **MyISAM** index files. Usually the size of an index block is equal to the size of nodes of the index B-tree. (Indexes are represented on disk using a B-tree data structure. Nodes at the bottom of the tree are leaf nodes. Nodes above the leaf nodes are non-leaf nodes.)

All block buffers in a key cache structure are the same size. This size can be equal to, greater than, or less than the size of a table index block. Usually one these two values is a multiple of the other.

When data from any table index block must be accessed, the server first checks whether it is available in some block buffer of the key cache. If it is, the server accesses data in the key cache rather than on disk. That is, it reads from the cache or writes into it rather than reading from or writing to disk. Otherwise, the server chooses a cache block buffer containing a different table index block (or blocks) and replaces the data there by a copy of required table index block. As soon as the new index block is in the cache, the index data can be accessed.

If it happens that a block selected for replacement has been modified, the block is considered “dirty.” In this case, prior to being replaced, its contents are flushed to the table index from which it came.

Usually the server follows an *LRU* (*Least Recently Used*) strategy: When choosing a block for replacement, it selects the least recently used index block. To make this choice easier, the key cache module maintains a special queue (*LRU chain*) of all used blocks. When a block is accessed, it is placed at the end of the queue. When blocks need to be replaced, blocks at the beginning of the queue are the least recently used and become the first candidates for eviction.

6.4.6.1. Shared Key Cache Access

Threads can access key cache buffers simultaneously, subject to the following conditions:

- A buffer that is not being updated can be accessed by multiple threads.
- A buffer that is being updated causes threads that need to use it to wait until the update is complete.
- Multiple threads can initiate requests that result in cache block replacements, as long as they do not interfere with each other (that is, as long as they need different index blocks, and thus cause different cache blocks to be replaced).

Shared access to the key cache enables the server to improve throughput significantly.

6.4.6.2. Multiple Key Caches

Shared access to the key cache improves performance but does not eliminate contention among threads entirely. They still compete for control structures that manage access to the key cache buffers. To reduce key cache access contention further, MySQL also provides multiple key caches. This feature enables you to assign different table indexes to different key caches.

Where there are multiple key caches, the server must know which cache to use when processing queries for a given [MyISAM](#) table. By default, all [MyISAM](#) table indexes are cached in the default key cache. To assign table indexes to a specific key cache, use the [CACHE INDEX](#) statement (see [Section 12.5.5.1, “CACHE INDEX Syntax”](#)). For example, the following statement assigns indexes from the tables [t1](#), [t2](#), and [t3](#) to the key cache named [hot_cache](#):

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
```

Table	Op	Msg_type	Msg_text
test.t1	assign_to_keycache	status	OK
test.t2	assign_to_keycache	status	OK
test.t3	assign_to_keycache	status	OK

The key cache referred to in a [CACHE INDEX](#) statement can be created by setting its size with a [SET GLOBAL](#) parameter setting statement or by using server startup options. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

To destroy a key cache, set its size to zero:

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

Note that you cannot destroy the default key cache. Any attempt to do this will be ignored:

```
mysql> SET GLOBAL key_buffer_size = 0;
```

```
mysql> SHOW VARIABLES LIKE 'key_buffer_size';
```

Variable_name	Value
key_buffer_size	8384512

Key cache variables are structured system variables that have a name and components. For [keycache1.key_buffer_size](#), [keycache1](#) is the cache variable name and [key_buffer_size](#) is the cache component. See [Section 5.2.4.1, “Structured System Variables”](#), for a description of the syntax used for referring to structured key cache system variables.

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

For a busy server, we recommend a strategy that uses three key caches:

- A “hot” key cache that takes up 20% of the space allocated for all key caches. Use this for tables that are heavily used for searches but that are not updated.
- A “cold” key cache that takes up 20% of the space allocated for all key caches. Use this cache for medium-sized, intensively modified tables, such as temporary tables.

- A “warm” key cache that takes up 60% of the key cache space. Employ this as the default key cache, to be used by default for all other tables.

One reason the use of three key caches is beneficial is that access to one key cache structure does not block access to the others. Statements that access tables assigned to one cache do not compete with statements that access tables assigned to another cache. Performance gains occur for other reasons as well:

- The hot cache is used only for retrieval queries, so its contents are never modified. Consequently, whenever an index block needs to be pulled in from disk, the contents of the cache block chosen for replacement need not be flushed first.
- For an index assigned to the hot cache, if there are no queries requiring an index scan, there is a high probability that the index blocks corresponding to non-leaf nodes of the index B-tree remain in the cache.
- An update operation most frequently executed for temporary tables is performed much faster when the updated node is in the cache and need not be read in from disk first. If the size of the indexes of the temporary tables are comparable with the size of cold key cache, the probability is very high that the updated node is in the cache.

CACHE INDEX sets up an association between a table and a key cache, but the association is lost each time the server restarts. If you want the association to take effect each time the server starts, one way to accomplish this is to use an option file: Include variable settings that configure your key caches, and an `init-file` option that names a file containing **CACHE INDEX** statements to be executed. For example:

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
init_file=/path/to/data-directory/mysql_init.sql
```

MySQL Enterprise

For advice on how best to configure your `my.cnf/my.ini` option file subscribe to MySQL Enterprise Monitor. Recommendations are based on actual table usage. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

The statements in `mysql_init.sql` are executed each time the server starts. The file should contain one SQL statement per line. The following example assigns several tables each to `hot_cache` and `cold_cache`:

```
CACHE INDEX db1.t1, db1.t2, db2.t3 IN hot_cache
CACHE INDEX db1.t4, db2.t5, db2.t6 IN cold_cache
```

6.4.6.3. Midpoint Insertion Strategy

By default, the key cache management system uses the LRU strategy for choosing key cache blocks to be evicted, but it also supports a more sophisticated method called the *midpoint insertion strategy*.

When using the midpoint insertion strategy, the LRU chain is divided into two parts: a hot sub-chain and a warm sub-chain. The division point between two parts is not fixed, but the key cache management system takes care that the warm part is not “too short,” always containing at least `key_cache_division_limit` percent of the key cache blocks. `key_cache_division_limit` is a component of structured key cache variables, so its value is a parameter that can be set per cache.

When an index block is read from a table into the key cache, it is placed at the end of the warm sub-chain. After a certain number of hits (accesses of the block), it is promoted to the hot sub-chain. At present, the number of hits required to promote a block (3) is the same for all index blocks.

A block promoted into the hot sub-chain is placed at the end of the chain. The block then circulates within this sub-chain. If the block stays at the beginning of the sub-chain for a long enough time, it is demoted to the warm chain. This time is determined by the value of the `key_cache_age_threshold` component of the key cache.

The threshold value prescribes that, for a key cache containing *N* blocks, the block at the beginning of the hot sub-chain not accessed within the last $N \times \text{key\_cache\_age\_threshold} / 100$ hits is to be moved to the beginning of the warm sub-chain. It then becomes the first candidate for eviction, because blocks for replacement always are taken from the beginning of the warm sub-chain.

The midpoint insertion strategy allows you to keep more-valued blocks always in the cache. If you prefer to use the plain LRU strategy, leave the `key_cache_division_limit` value set to its default of 100.

The midpoint insertion strategy helps to improve performance when execution of a query that requires an index scan effectively pushes out of the cache all the index blocks corresponding to valuable high-level B-tree nodes. To avoid this, you must use a midpoint insertion strategy with the `key_cache_division_limit` set to much less than 100. Then valuable frequently hit nodes are preserved in the hot sub-chain during an index scan operation as well.

6.4.6.4. Index Preloading

If there are enough blocks in a key cache to hold blocks of an entire index, or at least the blocks corresponding to its non-leaf nodes, it makes sense to preload the key cache with index blocks before starting to use it. Preloading allows you to put the table index blocks into a key cache buffer in the most efficient way: by reading the index blocks from disk sequentially.

Without preloading, the blocks are still placed into the key cache as needed by queries. Although the blocks will stay in the cache, because there are enough buffers for all of them, they are fetched from disk in random order, and not sequentially.

To preload an index into a cache, use the `LOAD INDEX INTO CACHE` statement. For example, the following statement preloads nodes (index blocks) of indexes of the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
```

Table	Op	Msg_type	Msg_text
test.t1	preload_keys	status	OK
test.t2	preload_keys	status	OK

The `IGNORE LEAVES` modifier causes only blocks for the non-leaf nodes of the index to be preloaded. Thus, the statement shown preloads all index blocks from `t1`, but only blocks for the non-leaf nodes from `t2`.

If an index has been assigned to a key cache using a `CACHE INDEX` statement, preloading places index blocks into that cache. Otherwise, the index is loaded into the default key cache.

6.4.6.5. Key Cache Block Size

It is possible to specify the size of the block buffers for an individual key cache using the `key_cache_block_size` variable. This permits tuning of the performance of I/O operations for index files.

The best performance for I/O operations is achieved when the size of read buffers is equal to the size of the native operating system I/O buffers. But setting the size of key nodes equal to the size of the I/O buffer does not always ensure the best overall performance. When reading the big leaf nodes, the server pulls in a lot of unnecessary data, effectively preventing reading other leaf nodes.

Currently, you cannot control the size of the index blocks in a table. This size is set by the server when the `.MYI` index file is created, depending on the size of the keys in the indexes present in the table definition. In most cases, it is set equal to the I/O buffer size.

6.4.6.6. Restructuring a Key Cache

A key cache can be restructured at any time by updating its parameter values. For example:

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

If you assign to either the `key_buffer_size` or `key_cache_block_size` key cache component a value that differs from the component's current value, the server destroys the cache's old structure and creates a new one based on the new values. If the cache contains any dirty blocks, the server saves them to disk before destroying and re-creating the cache. Restructuring does not occur if you change other key cache parameters.

When restructuring a key cache, the server first flushes the contents of any dirty buffers to disk. After that, the cache contents become unavailable. However, restructuring does not block queries that need to use indexes assigned to the cache. Instead, the server directly accesses the table indexes using native filesystem caching. Filesystem caching is not as efficient as using a key cache, so although queries execute, a slowdown can be anticipated. After the cache has been restructured, it becomes available again for caching indexes assigned to it, and the use of filesystem caching for the indexes ceases.

6.4.7. MyISAM Index Statistics Collection

Storage engines collect statistics about tables for use by the optimizer. Table statistics are based on value groups, where a value group is a set of rows with the same key prefix value. For optimizer purposes, an important statistic is the average value group size.

MySQL uses the average value group size in the following ways:

- To estimate how many rows must be read for each `ref` access
- To estimate how many row a partial join will produce; that is, the number of rows that an operation of this form will produce:

```
(...) JOIN tbl_name ON tbl_name.key = expr
```

As the average value group size for an index increases, the index is less useful for those two purposes because the average number of rows per lookup increases: For the index to be good for optimization purposes, it is best that each index value target a small number of rows in the table. When a given index value yields a large number of rows, the index is less useful and MySQL is less likely to use it.

The average value group size is related to table cardinality, which is the number of value groups. The `SHOW INDEX` statement displays a cardinality value based on N/S , where N is the number of rows in the table and S is the average value group size. That ratio yields an approximate number of value groups in the table.

For a join based on the `<=>` comparison operator, `NULL` is not treated differently from any other value: `NULL <=> NULL`, just as `N <=> N` for any other N .

However, for a join based on the `=` operator, `NULL` is different from non-`NULL` values: `expr1 = expr2` is not true when `expr1` or `expr2` (or both) are `NULL`. This affects `ref` accesses for comparisons of the form `tbl_name.key = expr`: MySQL will not access the table if the current value of `expr` is `NULL`, because the comparison cannot be true.

For `=` comparisons, it does not matter how many `NULL` values are in the table. For optimization purposes, the relevant value is the average size of the non-`NULL` value groups. However, MySQL does not currently allow that average size to be collected or used.

For `MyISAM` tables, you have some control over collection of table statistics by means of the `myisam_stats_method` system variable. This variable has two possible values, which differ as follows:

- When `myisam_stats_method` is `nulls_equal`, all `NULL` values are treated as identical (that is, they all form a single value group).

If the `NULL` value group size is much higher than the average non-`NULL` value group size, this method skews the average value group size upward. This makes index appear to the optimizer to be less useful than it really is for joins that look for non-`NULL` values. Consequently, the `nulls_equal` method may cause the optimizer not to use the index for `ref` accesses when it should.

- When `myisam_stats_method` is `nulls_unequal`, `NULL` values are not considered the same. Instead, each `NULL` value forms a separate value group of size 1.

If you have many `NULL` values, this method skews the average value group size downward. If the average non-`NULL` value group size is large, counting `NULL` values each as a group of size 1 causes the optimizer to overestimate the value of the index for joins that look for non-`NULL` values. Consequently, the `nulls_unequal` method may cause the optimizer to use this index for `ref` lookups when other methods may be better.

If you tend to use many joins that use `<=>` rather than `=`, `NULL` values are not special in comparisons and one `NULL` is equal to another. In this case, `nulls_equal` is the appropriate statistics method.

The `myisam_stats_method` system variable has global and session values. Setting the global value affects `MyISAM` statistics collection for all `MyISAM` tables. Setting the session value affects statistics collection only for the current client connection. This means that you can force a table's statistics to be regenerated with a given method without affecting other clients by setting the session value of `myisam_stats_method`.

To regenerate table statistics, you can use any of the following methods:

- Set `myisam_stats_method`, and then issue a `CHECK TABLE` statement
- Execute `myisamchk --stats_method=method_name --analyze`
- Change the table to cause its statistics to go out of date (for example, insert a row and then delete it), and then set `myisam_stats_method` and issue an `ANALYZE TABLE` statement

Some caveats regarding the use of `myisam_stats_method`:

- You can force table statistics to be collected explicitly, as just described. However, MySQL may also collect statistics automatically. For example, if during the course of executing statements for a table, some of those statements modify the table, MySQL may collect statistics. (This may occur for bulk inserts or deletes, or some `ALTER TABLE` statements, for example.) If this happens, the statistics are collected using whatever value `myisam_stats_method` has at the time. Thus, if you collect statistics using one method, but `myisam_stats_method` is set to the other method when a table's statistics are collected automatically later, the other method will be used.
- There is no way to tell which method was used to generate statistics for a given `MyISAM` table.
- `myisam_stats_method` applies only to `MyISAM` tables. Other storage engines have only one method for collecting table statistics. Usually it is closer to the `nulls_equal` method.

6.4.8. How MySQL Opens and Closes Tables

When you execute a `mysqladmin status` command, you should see something like this:

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

The `Open tables` value of 12 can be somewhat puzzling if you have only six tables.

MySQL is multi-threaded, so there may be many clients issuing queries for a given table simultaneously. To minimize the problem with multiple client threads having different states on the same table, the table is opened independently by each concurrent thread. This uses additional memory but normally increases performance. With `MyISAM` tables, one extra file descriptor is required for the data file for each client that has the table open. (By contrast, the index file descriptor is shared between all threads.)

Note

`table_open_cache` was previously known as `table_cache` in MySQL 5.1.2 and earlier.

The `table_open_cache`, `max_connections`, and `max_tmp_tables` system variables affect the maximum number of files the server keeps open. If you increase one or more of these values, you may run up against a limit imposed by your operating system on the per-process number of open file descriptors. Many operating systems allow you to increase the open-files limit, although the method varies widely from system to system. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so.

`table_open_cache` is related to `max_connections`. For example, for 200 concurrent running connections, you should have a table cache size of at least $200 \times N$, where N is the maximum number of tables per join in any of the queries which you execute. You must also reserve some extra file descriptors for temporary tables and files.

Make sure that your operating system can handle the number of open file descriptors implied by the `table_open_cache` setting. If `table_open_cache` is set too high, MySQL may run out of file descriptors and refuse connections, fail to perform queries, and be very unreliable. You also have to take into account that the `MyISAM` storage engine needs two file descriptors for each unique open table. You can increase the number of file descriptors available to MySQL using the `--open-files-limit` startup option to `mysqld`. See [Section B.1.2.17, “'FILE' NOT FOUND and Similar Errors”](#).

The cache of open tables is kept at a level of `table_open_cache` entries. The default value is 64; this can be changed with the `-table_open_cache` option to `mysqld`. Note that MySQL may temporarily open more tables than this to execute queries.

MySQL Enterprise

Performance may suffer if `table_cache` is set too low. For expert advice on the optimum value for this variable, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

MySQL closes an unused table and removes it from the table cache under the following circumstances:

- When the cache is full and a thread tries to open a table that is not in the cache.
- When the cache contains more than `table_open_cache` entries and a table in the cache is no longer being used by any threads.

- When a table flushing operation occurs. This happens when someone issues a `FLUSH TABLES` statement or executes a `mysqladmin flush-tables` or `mysqladmin refresh` command.

When the table cache fills up, the server uses the following procedure to locate a cache entry to use:

- Tables that are not currently in use are released, beginning with the table least recently used.
- If a new table needs to be opened, but the cache is full and no tables can be released, the cache is temporarily extended as necessary.

When the cache is in a temporarily extended state and a table goes from a used to unused state, the table is closed and released from the cache.

A table is opened for each concurrent access. This means the table needs to be opened twice if two threads access the same table or if a thread accesses the table twice in the same query (for example, by joining the table to itself). Each concurrent open requires an entry in the table cache. The first open of any `MyISAM` table takes two file descriptors: one for the data file and one for the index file. Each additional use of the table takes only one file descriptor for the data file. The index file descriptor is shared among all threads.

If you are opening a table with the `HANDLER tbl_name OPEN` statement, a dedicated table object is allocated for the thread. This table object is not shared by other threads and is not closed until the thread calls `HANDLER tbl_name CLOSE` or the thread terminates. When this happens, the table is put back in the table cache (if the cache is not full). See [Section 12.2.3, “HANDLER Syntax”](#).

You can determine whether your table cache is too small by checking the `mysqld` status variable `Opened_tables`:

```
mysql> SHOW GLOBAL STATUS LIKE 'Opened_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Opened_tables | 2741  |
+-----+-----+
```

If the value is very large, even when you have not issued many `FLUSH TABLES` statements, you should increase the table cache size. See [Section 5.2.3, “System Variables”](#), and [Section 5.2.5, “Status Variables”](#).

6.4.9. Drawbacks to Creating Many Tables in the Same Database

If you have many `MyISAM` tables in the same database directory, open, close, and create operations are slow. If you execute `SELECT` statements on many different tables, there is a little overhead when the table cache is full, because for every table that has to be opened, another must be closed. You can reduce this overhead by making the table cache larger.

6.5. Optimizing the MySQL Server

6.5.1. System Factors and Startup Parameter Tuning

We start with system-level factors, because some of these decisions must be made very early to achieve large performance gains. In other cases, a quick look at this section may suffice. However, it is always nice to have a sense of how much can be gained by changing factors that apply at this level.

The operating system to use is very important. To get the best use of multiple-CPU machines, you should use Solaris (because its threads implementation works well) or Linux (because the 2.4 and later kernels have good SMP support). Note that older Linux kernels have a 2GB filesize limit by default. If you have such a kernel and a need for files larger than 2GB, you should get the Large File Support (LFS) patch for the ext2 filesystem. Other filesystems such as ReiserFS and XFS do not have this 2GB limitation.

Before using MySQL in production, we advise you to test it on your intended platform.

Other tips:

- If you have enough RAM, you could remove all swap devices. Some operating systems use a swap device in some contexts even if you have free memory.
- Avoid external locking. Since MySQL 4.0, the default has been for external locking to be disabled on all systems. The `--external-locking` and `--skip-external-locking` options explicitly enable and disable external locking.

Note that disabling external locking does not affect MySQL's functionality as long as you run only one server. Just remember to take down the server (or lock and flush the relevant tables) before you run `myisamchk`. On some systems it is mandatory to disable external locking because it does not work, anyway.

The only case in which you cannot disable external locking is when you run multiple MySQL *servers* (not clients) on the same data, or if you run `myisamchk` to check (not repair) a table without telling the server to flush and lock the tables first. Note that using multiple MySQL servers to access the same data concurrently is generally *not* recommended, except when using MySQL Cluster.

The `LOCK TABLES` and `UNLOCK TABLES` statements use internal locking, so you can use them even if external locking is disabled.

6.5.2. Tuning Server Parameters

You can determine the default buffer sizes used by the `mysqld` server using this command:

```
shell> mysqld --verbose --help
```

This command produces a list of all `mysqld` options and configurable system variables. The output includes the default variable values and looks something like this:

```
help                TRUE
abort-slave-event-count 0
allow-suspicious-udfs  FALSE
auto-increment-increment 1
auto-increment-offset 1
automatic-sp-privileges TRUE
basedir              /home/mysql/
bind-address          (No default value)
character-set-client-handshake TRUE
character-set-server  latin1
character-sets-dir     /home/mysql/share/mysql/charsets/
chroot                (No default value)
collation-server       latin1_swedish_ci
completion-type        0
concurrent-insert      1
console               FALSE
datadir               /home/mysql/var/
default-character-set  latin1
default-collation      latin1_swedish_ci
default-time-zone      (No default value)
disconnect-slave-event-count 0
enable-locking         FALSE
enable-pstack          FALSE
engine-condition-pushdown FALSE
external-locking       FALSE
gdb                   FALSE
large-pages            FALSE
init-connect           (No default value)
init-file              (No default value)
init-slave              (No default value)
innodb                 TRUE
innodb_checksums       TRUE
innodb_data_home_dir   (No default value)
innodb_doublewrite     TRUE
innodb_fast_shutdown  1
innodb_file_per_table  FALSE
innodb_flush_log_at_trx_commit 1
innodb_flush_method    (No default value)
innodb_locks_unsafe_for_binlog FALSE
innodb_log_arch_dir    (No default value)
innodb_log_group_home_dir (No default value)
innodb_max_dirty_pages_pct 90
innodb_max_purge_lag   0
innodb_status_file     FALSE
innodb_table_locks     TRUE
innodb_support_xa      TRUE
isam                   FALSE
language               /home/mysql/share/mysql/english
local-infile           TRUE
log                    /home/mysql/var/master1.log
log-bin                 /home/mysql/var/master1
log-bin-index           (No default value)
log-bin-trust-routine-creators FALSE
log-error               /home/mysql/var/master1.err
log-isam                myisam.log
log-queries-not-using-indexes FALSE
log-short-format        FALSE
log-slave-updates       FALSE
log-slow-admin-statements FALSE
```

log-slow-queries	(No default value)
log-tc	tc.log
log-tc-size	24576
log-update	(No default value)
log-warnings	1
low-priority-updates	FALSE
master-connect-retry	60
master-host	(No default value)
master-info-file	master.info
master-password	(No default value)
master-port	3306
master-retry-count	86400
master-ssl	FALSE
master-ssl-ca	(No default value)
master-ssl-capath	(No default value)
master-ssl-cert	(No default value)
master-ssl-cipher	(No default value)
master-ssl-key	(No default value)
master-user	test
max-binlog-dump-events	0
memlock	FALSE
myisam-recover	OFF
ndbcluster	FALSE
ndb-connectstring	(No default value)
ndb-mgmd-host	(No default value)
ndb-nodeid	0
ndb-autoincrement-prefetch-sz	32
ndb-distribution	KEYHASH
ndb-force-send	TRUE
ndb_force_send	TRUE
ndb-use-exact-count	TRUE
ndb_use_exact_count	TRUE
ndb-shm	FALSE
ndb-optimized-node-selection	TRUE
ndb-cache-check-time	0
ndb-index-stat-enable	TRUE
ndb-index-stat-cache-entries	32
ndb-index-stat-update-freq	20
new	FALSE
old-alter-table	FALSE
old-passwords	FALSE
old-style-user-limits	FALSE
pid-file	/home/mysql/var/hostname.pid1
port	3306
relay-log	(No default value)
relay-log-index	(No default value)
relay-log-info-file	relay-log.info
replicate-same-server-id	FALSE
report-host	(No default value)
report-password	(No default value)
report-port	3306
report-user	(No default value)
rpl-recovery-rank	0
safe-user-create	FALSE
secure-auth	FALSE
server-id	1
show-slave-auth-info	FALSE
skip-grant-tables	FALSE
skip-slave-start	FALSE
slave-load-tmpdir	/tmp/
socket	/tmp/mysql.sock
sporadic-binlog-dump-fail	FALSE
sql-mode	OFF
symbolic-links	TRUE
tc-heuristic-recover	(No default value)
temp-pool	TRUE
timed_mutexes	FALSE
tmpdir	(No default value)
use-symbolic-links	TRUE
verbose	TRUE
warnings	1
back_log	50
binlog_cache_size	32768
bulk_insert_buffer_size	8388608
connect_timeout	5
date_format	(No default value)
datetime_format	(No default value)
default_week_format	0
delayed_insert_limit	100
delayed_insert_timeout	300
delayed_queue_size	1000
expire_logs_days	0
flush_time	0
ft_max_word_len	84
ft_min_word_len	4
ft_query_expansion_limit	20
ft_stopword_file	(No default value)
group_concat_max_len	1024
innodb_additional_mem_pool_size	1048576
innodb_autoextend_increment	8

innodb_buffer_pool_ave_mem_mb	0
innodb_buffer_pool_size	8388608
innodb_concurrency_tickets	500
innodb_file_io_threads	4
innodb_force_recovery	0
innodb_lock_wait_timeout	50
innodb_log_buffer_size	1048576
innodb_log_file_size	5242880
innodb_log_files_in_group	2
innodb_mirrored_log_groups	1
innodb_open_files	300
innodb_sync_spin_loops	20
innodb_thread_concurrency	20
innodb_commit_concurrency	0
innodb_thread_sleep_delay	10000
interactive_timeout	28800
join_buffer_size	131072
key_buffer_size	8388600
key_cache_age_threshold	300
key_cache_block_size	1024
key_cache_division_limit	100
long_query_time	10
lower_case_table_names	0
max_allowed_packet	1048576
max_binlog_cache_size	4294967295
max_binlog_size	1073741824
max_connect_errors	10
max_connections	100
max_delayed_threads	20
max_error_count	64
max_heap_table_size	16777216
max_join_size	4294967295
max_length_for_sort_data	1024
max_relay_log_size	0
max_seeks_for_key	4294967295
max_sort_length	1024
max_tmp_tables	32
max_user_connections	0
max_write_lock_count	4294967295
multi_range_count	256
myisam_block_size	1024
myisam_data_pointer_size	6
myisam_max_extra_sort_file_size	2147483648
myisam_max_sort_file_size	2147483647
myisam_repair_threads	1
myisam_sort_buffer_size	8388608
myisam_stats_method	nulls_unequal
net_buffer_length	16384
net_read_timeout	30
net_retry_count	10
net_write_timeout	60
open_files_limit	0
optimizer_prune_level	1
optimizer_search_depth	62
preload_buffer_size	32768
query_alloc_block_size	8192
query_cache_limit	1048576
query_cache_min_res_unit	4096
query_cache_size	0
query_cache_type	1
query_cache_wlock_invalidate	FALSE
query_prealloc_size	8192
range_alloc_block_size	2048
read_buffer_size	131072
read_only	FALSE
read_rnd_buffer_size	262144
div_precision_increment	4
record_buffer	131072
relay_log_purge	TRUE
relay_log_space_limit	0
slave_compressed_protocol	FALSE
slave_net_timeout	3600
slave_transaction_retries	10
slow_launch_time	2
sort_buffer_size	2097144
sync_binlog	0
sync_frm	TRUE
sync-replication	0
sync-replication-slave-id	0
sync-replication-timeout	10
table_open_cache	64
table_lock_wait_timeout	50
thread_cache_size	0
thread_concurrency	10
thread_stack	196608
time_format	(No default value)
tmp_table_size	33554432
transaction_alloc_block_size	8192
transaction_prealloc_size	4096
updatable_views_with_limit	1

```
wait_timeout      28800
```

For a `mysqld` server that is currently running, you can see the current values of its system variables by connecting to it and issuing this statement:

```
mysql> SHOW VARIABLES;
```

You can also see some statistical and status indicators for a running server by issuing this statement:

```
mysql> SHOW STATUS;
```

System variable and status information also can be obtained using `mysqladmin`:

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

For a full description of all system and status variables, see [Section 5.2.3, “System Variables”](#), and [Section 5.2.5, “Status Variables”](#).

MySQL uses algorithms that are very scalable, so you can usually run with very little memory. However, normally you get better performance by giving MySQL more memory.

When tuning a MySQL server, the two most important variables to configure are `key_buffer_size` and `table_open_cache`. You should first feel confident that you have these set appropriately before trying to change any other variables.

The following examples indicate some typical variable values for different runtime configurations.

- If you have at least 256MB of memory and many tables and want maximum performance with a moderate number of clients, you should use something like this:

```
shell> mysqld_safe --key_buffer_size=64M --table_open_cache=256 \
               --sort_buffer_size=4M --read_buffer_size=1M &
```

- If you have only 128MB of memory and only a few tables, but you still do a lot of sorting, you can use something like this:

```
shell> mysqld_safe --key_buffer_size=16M --sort_buffer_size=1M
```

If there are very many simultaneous connections, swapping problems may occur unless `mysqld` has been configured to use very little memory for each connection. `mysqld` performs better if you have enough memory for all connections.

- With little memory and lots of connections, use something like this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \
               --read_buffer_size=100K &
```

Or even this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \
               --table_open_cache=32 --read_buffer_size=8K \
               --net_buffer_length=1K &
```

If you are performing `GROUP BY` or `ORDER BY` operations on tables that are much larger than your available memory, you should increase the value of `read_rnd_buffer_size` to speed up the reading of rows following sorting operations.

You can make use of the example option files included with your MySQL distribution; see [Section 4.3.2.1, “Preconfigured Option Files”](#).

If you specify an option on the command line for `mysqld` or `mysqld_safe`, it remains in effect only for that invocation of the server. To use the option every time the server runs, put it in an option file.

To see the effects of a parameter change, do something like this:

```
shell> mysqld --key_buffer_size=32M --verbose --help
```

The variable values are listed near the end of the output. Make sure that the `--verbose` and `--help` options are last. Otherwise, the effect of any options listed after them on the command line are not reflected in the output.

For information on tuning the InnoDB storage engine, see [Section 13.5.11, “InnoDB Performance Tuning Tips”](#).

MySQL Enterprise

For expert advice on tuning system parameters subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

6.5.3. Controlling Query Optimizer Performance

The task of the query optimizer is to find an optimal plan for executing an SQL query. Because the difference in performance between “good” and “bad” plans can be orders of magnitude (that is, seconds versus hours or even days), most query optimizers, including that of MySQL, perform a more or less exhaustive search for an optimal plan among all possible query evaluation plans. For join queries, the number of possible plans investigated by the MySQL optimizer grows exponentially with the number of tables referenced in a query. For small numbers of tables (typically less than 7–10) this is not a problem. However, when larger queries are submitted, the time spent in query optimization may easily become the major bottleneck in the server’s performance.

A more flexible method for query optimization allows the user to control how exhaustive the optimizer is in its search for an optimal query evaluation plan. The general idea is that the fewer plans that are investigated by the optimizer, the less time it spends in compiling a query. On the other hand, because the optimizer skips some plans, it may miss finding an optimal plan.

The behavior of the optimizer with respect to the number of plans it evaluates can be controlled via two system variables:

- The `optimizer_prune_level` variable tells the optimizer to skip certain plans based on estimates of the number of rows accessed for each table. Our experience shows that this kind of “educated guess” rarely misses optimal plans, and may dramatically reduce query compilation times. That is why this option is on (`optimizer_prune_level=1`) by default. However, if you believe that the optimizer missed a better query plan, this option can be switched off (`optimizer_prune_level=0`) with the risk that query compilation may take much longer. Note that, even with the use of this heuristic, the optimizer still explores a roughly exponential number of plans.
- The `optimizer_search_depth` variable tells how far into the “future” of each incomplete plan the optimizer should look to evaluate whether it should be expanded further. Smaller values of `optimizer_search_depth` may result in orders of magnitude smaller query compilation times. For example, queries with 12, 13, or more tables may easily require hours and even days to compile if `optimizer_search_depth` is close to the number of tables in the query. At the same time, if compiled with `optimizer_search_depth` equal to 3 or 4, the optimizer may compile in less than a minute for the same query. If you are unsure of what a reasonable value is for `optimizer_search_depth`, this variable can be set to 0 to tell the optimizer to determine the value automatically.

6.5.4. The MySQL Query Cache

The query cache stores the text of a `SELECT` statement together with the corresponding result that was sent to the client. If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again.

The query cache is extremely useful in an environment where you have tables that do not change very often and for which the server receives many identical queries. This is a typical situation for many Web servers that generate many dynamic pages based on database content.

Note

The query cache does not return stale data. When tables are modified, any relevant entries in the query cache are flushed.

The query cache does not work in an environment where you have multiple `mysqld` servers updating the same `MyISAM` tables.

Note

The query cache is not used for server-side prepared statements. If you’re using server-side prepared statements consider that these statement won’t be satisfied by the query cache. See [Section 25.2.4, “C API Prepared Statements”](#).

Some performance data for the query cache follows. These results were generated by running the MySQL benchmark suite on a Linux Alpha 2×500MHz system with 2GB RAM and a 64MB query cache.

- If all the queries you are performing are simple (such as selecting a row from a table with one row), but still differ so that the queries cannot be cached, the overhead for having the query cache active is 13%. This could be regarded as the worst case scenario. In real life, queries tend to be much more complicated, so the overhead normally is significantly lower.
- Searches for a single row in a single-row table are 238% faster with the query cache than without it. This can be regarded as close to the minimum speedup to be expected for a query that is cached.

To disable the query cache at server startup, set the `query_cache_size` system variable to 0. By disabling the query cache code, there is no noticeable overhead. If you build MySQL from source, query cache capabilities can be excluded from the server entirely by invoking `configure` with the `--without-query-cache` option.

6.5.4.1. How the Query Cache Operates

This section describes how the query cache works when it is operational. [Section 6.5.4.3, “Query Cache Configuration”](#), describes how to control whether it is operational.

Incoming queries are compared to those in the query cache before parsing, so the following two queries are regarded as different by the query cache:

```
SELECT * FROM tbl_name
Select * from tbl_name
```

Queries must be *exactly* the same (byte for byte) to be seen as identical. In addition, query strings that are identical may be treated as different for other reasons. Queries that use different databases, different protocol versions, or different default character sets are considered different queries and are cached separately.

The cache is not used for queries of the following types:

- Queries that are a subquery of an outer query
- Queries executed within the body of a stored procedure, stored function, trigger, or event

Before a query result is fetched from the query cache, MySQL checks that the user has `SELECT` privilege for all databases and tables involved. If this is not the case, the cached result is not used.

If a query result is returned from query cache, the server increments the `Qcache_hits` status variable, not `Com_select`. See [Section 6.5.4.4, “Query Cache Status and Maintenance”](#).

If a table changes, all cached queries that use the table become invalid and are removed from the cache. This includes queries that use `MERGE` tables that map to the changed table. A table can be changed by many types of statements, such as `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `ALTER TABLE`, `DROP TABLE`, or `DROP DATABASE`.

The query cache also works within transactions when using `InnoDB` tables.

In MySQL 5.1, queries generated by views are cached.

The query cache works for `SELECT SQL_CALC_FOUND_ROWS . . .` queries and stores a value that is returned by a following `SELECT FOUND_ROWS( )` query. `FOUND_ROWS( )` returns the correct value even if the preceding query was fetched from the cache because the number of found rows is also stored in the cache. The `SELECT FOUND_ROWS( )` query itself cannot be cached.

Before MySQL 5.1.17, prepared statements do not use the query cache. Beginning with 5.1.17, prepared statements use the query cache under certain conditions, which differ depending on the preparation method:

- Statements that are issued via the binary protocol using `mysql_stmt_prepare( )` and `mysql_stmt_execute( )`. See [Section 25.2.4, “C API Prepared Statements”](#).

For a prepared statement executed via the binary protocol, comparison with statements in the query cache is based on the text of the statement after expansion of `?` parameter markers. The statement is compared only with other cached statements that were executed via the binary protocol. That is, for query cache purposes, statements issued via the binary protocol are distinct from statements issued via the text protocol.

- Statements that are issued via the text (non-binary) protocol using `PREPARE` and `EXECUTE`. See [Section 12.7, “SQL Syntax for](#)

Prepared Statements". These are denoted SQL PS statements here.

Before MySQL 5.1.21, for a prepared statement executed via **PREPARE** and **EXECUTE**, it is not cached if it contains any **?** parameter markers. In that case, the statement after parameter expansion contains references to user variables, which prevents caching, even for non-prepared statements. If the statement contains no parameter markers, the statement is compared with statements in the query cache that were executed via the text protocol (that is, it is compared with other SQL PS statements and non-prepared statements). As of MySQL 5.1.21, this limitation is lifted and prepared statements that contain parameter markers can be cached because expansion directly substitutes the user variable values.

A query cannot be cached if it contains any of the functions shown in the following table:

BENCHMARK()	CONNECTION_ID()	CURDATE()
CURRENT_DATE()	CURRENT_TIME()	CURRENT_TIMESTAMP()
CURTIME()	DATABASE()	ENCRYPT() with one parameter
FOUND_ROWS()	GET_LOCK()	LAST_INSERT_ID()
LOAD_FILE()	MASTER_POS_WAIT()	NOW()
RAND()	RELEASE_LOCK()	SLEEP()
SYSDATE()	UNIX_TIMESTAMP() with no parameters	USER()

A query also is not cached under these conditions:

- It refers to user-defined functions (UDFs) or stored functions.
- It refers to user variables.
- It refers to tables in the **mysql** system database.
- It is of any of the following forms:

```
SELECT ... IN SHARE MODE
SELECT ... FOR UPDATE
SELECT ... INTO OUTFILE ...
SELECT ... INTO DUMPFILE ...
SELECT * FROM ... WHERE autoincrement_col IS NULL
```

The last form is not cached because it is used as the ODBC workaround for obtaining the last insert ID value. See the MyODBC section of [Chapter 26, Connectors](#).

- It uses **TEMPORARY** tables.
- It does not use any tables.
- The user has a column-level privilege for any of the involved tables.

6.5.4.2. Query Cache **SELECT** Options

Two query cache-related options may be specified in **SELECT** statements:

- **SQL_CACHE**

The query result is cached if the value of the **query_cache_type** system variable is **ON** or **DEMAND**.

- **SQL_NO_CACHE**

The query result is not cached.

Examples:

```
SELECT SQL_CACHE id, name FROM customer;
SELECT SQL_NO_CACHE id, name FROM customer;
```

6.5.4.3. Query Cache Configuration

The `have_query_cache` server system variable indicates whether the query cache is available:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES  |
+-----+-----+
```

When using a standard MySQL binary, this value is always **YES**, even if query caching is disabled.

Several other system variables control query cache operation. These can be set in an option file or on the command line when starting `mysqld`. The query cache system variables all have names that begin with `query_cache_`. They are described briefly in [Section 5.2.3, “System Variables”](#), with additional configuration information given here.

To set the size of the query cache, set the `query_cache_size` system variable. Setting it to 0 disables the query cache. The default size is 0, so the query cache is disabled by default.

MySQL Enterprise

For expert advice on configuring the query cache subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

Note

When using the Windows Configuration Wizard to install or configure MySQL, the default value for `query_cache_size` will be configured automatically for you based on the different configuration types available. When using the Windows Configuration Wizard, the query cache may be enabled (i.e. set to a non-zero value) due to the selected configuration. The query cache is also controlled by the setting of the `query_cache_type` variable. You should check the values of these variables as set in your `my.ini` file after configuration has taken place.

When you set `query_cache_size` to a non-zero value, keep in mind that the query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value too small, you'll get a warning, as in this example:

```
mysql> SET GLOBAL query_cache_size = 40000;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1282
Message: Query cache failed to set size 39936; »
        new query cache size is 0

mysql> SET GLOBAL query_cache_size = 41984;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| query_cache_size | 41984 |
+-----+-----+
```

For the query cache to actually be able to hold any query results, its size must be set larger:

```
mysql> SET GLOBAL query_cache_size = 1000000;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| query_cache_size | 999424 |
+-----+-----+
1 row in set (0.00 sec)
```

The `query_cache_size` is aligned to the nearest 1024 byte block. The value reported may therefore be different from the value that you set.

If the query cache size is greater than 0, the `query_cache_type` variable influences how it works. This variable can be set to the following values:

- A value of `0` or `OFF` prevents caching or retrieval of cached results.
- A value of `1` or `ON` allows caching except of those statements that begin with `SELECT SQL_NO_CACHE`.
- A value of `2` or `DEMAND` causes caching of only those statements that begin with `SELECT SQL_CACHE`.

Setting the `GLOBAL query_cache_type` value determines query cache behavior for all clients that connect after the change is made. Individual clients can control cache behavior for their own connection by setting the `SESSION query_cache_type` value. For example, a client can disable use of the query cache for its own queries like this:

```
mysql> SET SESSION query_cache_type = OFF;
```

To control the maximum size of individual query results that can be cached, set the `query_cache_limit` system variable. The default value is 1MB.

Note

You can set the maximum size that can be specified for the query cache during runtime with the `SET` statement by using the `--maximum-query_cache_size=32M` option on the command line or in the configuration file.

When a query is to be cached, its result (the data sent to the client) is stored in the query cache during result retrieval. Therefore the data usually is not handled in one big chunk. The query cache allocates blocks for storing this data on demand, so when one block is filled, a new block is allocated. Because memory allocation operation is costly (timewise), the query cache allocates blocks with a minimum size given by the `query_cache_min_res_unit` system variable. When a query is executed, the last result block is trimmed to the actual data size so that unused memory is freed. Depending on the types of queries your server executes, you might find it helpful to tune the value of `query_cache_min_res_unit`:

- The default value of `query_cache_min_res_unit` is 4KB. This should be adequate for most cases.
- If you have a lot of queries with small results, the default block size may lead to memory fragmentation, as indicated by a large number of free blocks. Fragmentation can force the query cache to prune (delete) queries from the cache due to lack of memory. In this case, you should decrease the value of `query_cache_min_res_unit`. The number of free blocks and queries removed due to pruning are given by the values of the `Qcache_free_blocks` and `Qcache_lowmem_prunes` status variables.
- If most of your queries have large results (check the `Qcache_total_blocks` and `Qcache_queries_in_cache` status variables), you can increase performance by increasing `query_cache_min_res_unit`. However, be careful to not make it too large (see the previous item).

MySQL Enterprise

If the query cache is under-utilized, performance will suffer. Advice on avoiding this problem is provided to subscribers to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

6.5.4.4. Query Cache Status and Maintenance

You can check whether the query cache is present in your MySQL server using the following statement:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES   |
+-----+-----+
```

You can defragment the query cache to better utilize its memory with the `FLUSH QUERY CACHE` statement. The statement does not remove any queries from the cache.

The **RESET QUERY CACHE** statement removes all query results from the query cache. The **FLUSH TABLES** statement also does this.

To monitor query cache performance, use **SHOW STATUS** to view the cache status variables:

```
mysql> SHOW STATUS LIKE 'Qcache%';
```

Variable_name	Value
Qcache_free_blocks	36
Qcache_free_memory	138488
Qcache_hits	79570
Qcache_inserts	27087
Qcache_lowmem_prunes	3114
Qcache_not_cached	22989
Qcache_queries_in_cache	415
Qcache_total_blocks	912

Descriptions of each of these variables are given in [Section 5.2.5, “Status Variables”](#). Some uses for them are described here.

The total number of **SELECT** queries is given by this formula:

```
Com_select
+ Qcache_hits
+ queries with errors found by parser
```

The **Com_select** value is given by this formula:

```
Qcache_inserts
+ Qcache_not_cached
+ queries with errors found during the column-privileges check
```

The query cache uses variable-length blocks, so **Qcache_total_blocks** and **Qcache_free_blocks** may indicate query cache memory fragmentation. After **FLUSH QUERY CACHE**, only a single free block remains.

Every cached query requires a minimum of two blocks (one for the query text and one or more for the query results). Also, every table that is used by a query requires one block. However, if two or more queries use the same table, only one table block needs to be allocated.

The information provided by the **Qcache_lowmem_prunes** status variable can help you tune the query cache size. It counts the number of queries that have been removed from the cache to free up memory for caching new queries. The query cache uses a least recently used (LRU) strategy to decide which queries to remove from the cache. Tuning information is given in [Section 6.5.4.3, “Query Cache Configuration”](#).

6.5.5. Examining Thread Information

When you are attempting to ascertain what your MySQL server is doing, it can be helpful to examine the process list, which is the set of threads currently executing within the server. Process list information is available from these sources:

- The **SHOW [FULL] PROCESSLIST** statement ([Section 12.5.4.25, “SHOW PROCESSLIST Syntax”](#))
- The **INFORMATION_SCHEMA PROCESSLIST** table ([Section 23.22, “The INFORMATION_SCHEMA PROCESSLIST Table”](#))
- The **mysqladmin processlist** command ([Section 7.10, “mysqladmin — Client for Administering a MySQL Server”](#))

You can always view information about your own threads. To view information about threads being executed for other accounts, you must have the **PROCESS** privilege.

Each process list entry contains several pieces of information:

- **Id** is the connection identifier for the client associated with the thread.
- **User** and **Host** indicate the account associated with the thread.
- **db** is the default database for the thread, or **NULL** if none is selected.

- **Command** and **State** indicate what the thread is doing.

Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

- **Time** indicates how long the thread has been in its current state.
- **Info** contains the text of the statement being executed by the thread, or **NULL** if it is not executing one. By default, this value contains only the first 100 characters of the statement. To see the complete statements, use **SHOW FULL PROCESSLIST**.

The following sections list the possible **Command** values, and **State** values grouped by category. The meaning for some of these values is self-evident. For others, additional description is provided. See also [Section 15.4.1, “Replication Implementation Details”](#), for additional information about thread states for replication servers.

6.5.5.1. Thread Command Values

A thread can have any of the following **Command** values:

- **Binlog Dump**

This is a thread on a master server for sending binary log contents to a slave server.

- **Change user**

The thread is executing a change-user operation.

- **Close stmt**

The thread is closing a prepared statement.

- **Connect**

A replication slave is connected to its master.

- **Connect Out**

A replication slave is connecting to its master.

- **Create DB**

The thread is executing a create-database operation.

- **Daemon**

This thread is internal to the server, not a thread that services a client connection.

- **Debug**

The thread is generating debugging information.

- **Delayed insert**

The thread is a delayed-insert handler.

- **Drop DB**

The thread is executing a drop-database operation.

- **Error**

- **Execute**

The thread is executing a prepared statement.

- **Fetch**

The thread is fetching the results from executing a prepared statement.

- **Field List**

The thread is retrieving information for table columns.

- **Init DB**

The thread is selecting a default database.

- **Kill**

The thread is killing another thread.

- **Long Data**

The thread is retrieving long data in the result of executing a prepared statement.

- **Ping**

The thread is handling a server-ping request.

- **Prepare**

The thread is preparing a prepared statement.

- **Processlist**

The thread is producing information about server threads.

- **Query**

The thread is executing a statement.

- **Quit**

The thread is terminating.

- **Refresh**

The thread is flushing table, logs, or caches, or resetting status variable or replication server information.

- **Register Slave**

The thread is registering a slave server.

- **Reset stmt**

The thread is resetting a prepared statement.

- **Set option**

The thread is setting or resetting a client statement-execution option.

- **Shutdown**

The thread is shutting down the server.

- **Sleep**

The thread is waiting for the client to send a new statement to it.

- **Statistics**

The thread is producing server-status information.

- **Table Dump**

The thread is sending table contents to a slave server.

- **Time**

Unused.

6.5.5.2. General Thread States

The following list describes thread **State** values that are associated with general query processing and not more specialized activities such as replication. Many of these are useful only for finding bugs in the server.

- **After create**

Occurs when the thread creates a table (including internal temporary tables), at the end of the function that creates the table. This state is used even if the table could not be created due to some error.

- **Analyzing**

The thread is calculating a **MyISAM** table key distributions (for example, for **ANALYZE TABLE**).

- **Checking table**

The thread is performing a table check operation.

- **cleaning up**

The thread has processed one command and is preparing to free memory and reset certain state variables.

- **closing tables**

Means that the thread is flushing the changed table data to disk and closing the used tables. This should be a fast operation. If not, you should verify that you do not have a full disk and that the disk is not in very heavy use.

- **converting HEAP to MyISAM**

The thread is converting an internal temporary table from a **MEMORY** table to an on-disk **MyISAM** table.

- **copy to tmp table**

The thread is processing an **ALTER TABLE** statement. This state occurs after the table with the new structure has been created but before rows are copied into it.

- **Copying to group table**

If a statement has different **ORDER BY** and **GROUP BY** criteria, the rows are sorted by group and copied to a temporary table.

- **Copying to tmp table**

The server is copying to a temporary table in memory.

- **Copying to tmp table on disk**

The server is copying to a temporary table on disk. The temporary result set was larger than **tmp_table_size** and the thread is changing the temporary table from in-memory to disk-based format to save memory.

- **Creating index**

The thread is processing **ALTER TABLE ... ENABLE KEYS** for a **MyISAM** table.

- **Creating sort index**

The thread is processing a **SELECT** that is resolved using an internal temporary table.

- `creating table`

The thread is creating a table. This includes creation of temporary tables.

- `Creating tmp table`

The thread is creating a temporary table in memory or on disk. If the table is created in memory but later is converted to an on-disk table, the state during that operation will be `Copying to tmp table on disk`.

- `deleting from main table`

The server is executing the first part of a multiple-table delete. It is deleting only from the first table, and saving fields and offsets to be used for deleting from the other (reference) tables.

- `deleting from reference tables`

The server is executing the second part of a multiple-table delete and deleting the matched rows from the other tables.

- `discard_or_import_tablespace`

The thread is processing an `ALTER TABLE ... DISCARD TABLESPACE` or `ALTER TABLE ... IMPORT TABLESPACE` statement.

- `end`

This occurs at the end but before the cleanup of `ALTER TABLE`, `CREATE VIEW`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements.

- `Execution of init_command`

The thread is executing statements in the value of the `init_command` system variable.

- `freeing items`

The thread has executed a command. This state is usually followed by `cleaning up`.

- `Flushing tables`

The thread is executing `FLUSH TABLES` and is waiting for all threads to close their tables.

- `FULLTEXT initialization`

The server is preparing to perform a natural-language full-text search.

- `init`

This occurs before the initialization of `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements.

- `Killed`

Someone has sent a `KILL` statement to the thread and it should abort next time it checks the kill flag. The flag is checked in each major loop in MySQL, but in some cases it might still take a short time for the thread to die. If the thread is locked by some other thread, the kill takes effect as soon as the other thread releases its lock.

- `Locked`

The query is locked by another query.

- `logging slow query`

The thread is writing a statement to the slow-query log.

- `login`

The initial state for a connection thread until the client has been authenticated successfully.

- `Opening tables,Opening table`

The thread is trying to open a table. This should be very fast procedure, unless something prevents opening. For example, an [ALTER TABLE](#) or a [LOCK TABLE](#) statement can prevent opening a table until the statement is finished.

- [preparing](#)

This state occurs during query optimization.

- [Purging old relay logs](#)

The thread is removing unneeded relay log files.

- [query end](#)

This state occurs after processing a query but before the [freeing items](#) state.

- [Reading from net](#)

The server is reading a packet from the network.

- [Removing duplicates](#)

The query was using [SELECT DISTINCT](#) in such a way that MySQL could not optimize away the distinct operation at an early stage. Because of this, MySQL requires an extra stage to remove all duplicated rows before sending the result to the client.

- [removing tmp table](#)

The thread is removing an internal temporary table after processing a [SELECT](#) statement. This state is not used if no temporary table was created.

- [rename](#)

The thread is renaming a table.

- [rename result table](#)

The thread is processing an [ALTER TABLE](#) statement, has created the new table, and is renaming it to replace the original table.

- [Reopen tables](#)

The thread got a lock for the table, but noticed after getting the lock that the underlying table structure changed. It has freed the lock, closed the table, and is trying to reopen it.

- [Repair by sorting](#)

The repair code is using a sort to create indexes.

- [Repair done](#)

The thread has completed a multi-threaded repair for a [MyISAM](#) table.

- [Repair with keycache](#)

The repair code is using creating keys one by one through the key cache. This is much slower than [Repair by sorting](#).

- [Rolling back](#)

The thread is rolling back a transaction.

- [Saving state](#)

For [MyISAM](#) table operations such as repair or analysis, the thread is saving the new table state to the [.MYI](#) file header. State includes information such as number of rows, the [AUTO_INCREMENT](#) counter, and key distributions.

- [Searching rows for update](#)

The thread is doing a first phase to find all matching rows before updating them. This has to be done if the [UPDATE](#) is changing the

index that is used to find the involved rows.

- **Sending data**

The thread is processing rows for a **SELECT** statement and also is sending data to the client.

- **setup**

The thread is beginning an **ALTER TABLE** operation.

- **Sorting for group**

The thread is doing a sort to satisfy a **GROUP BY**.

- **Sorting for order**

The thread is doing a sort to satisfy a **ORDER BY**.

- **Sorting index**

The thread is sorting index pages for more efficient access during a **MyISAM** table optimization operation.

- **Sorting result**

For a **SELECT** statement, this is similar to **Creating sort index**, but for non-temporary tables.

- **statistics**

The server is calculating statistics to develop a query execution plan.

- **System lock**

The thread is going to request or is waiting for an external system lock for the table. If you are not using multiple **mysqld** servers that are accessing the same tables, you can disable system locks with the **--skip-external-locking** option.

- **Table lock**

The next thread state after **System lock**. The thread has acquired an external lock and is going to request an internal table lock.

- **Updating**

The thread is searching for rows to update and is updating them.

- **updating main table**

The server is executing the first part of a multiple-table update. It is updating only the first table, and saving fields and offsets to be used for updating the other (reference) tables.

- **updating reference tables**

The server is executing the second part of a multiple-table update and updating the matched rows from the other tables.

- **User lock**

The thread is going to request or is waiting for an advisory lock requested with a **GET_LOCK()** call.

- **Waiting for tables,Waiting for table**

The thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to reopen the table, it must wait until all other threads have closed the table in question.

This notification takes place if another thread has used **FLUSH TABLES** or one of the following statements on the table in question: **FLUSH TABLES tbl_name**, **ALTER TABLE**, **RENAME TABLE**, **REPAIR TABLE**, **ANALYZE TABLE**, or **OPTIMIZE TABLE**.

- **Waiting on cond**

A generic state in which the thread is waiting for a condition to become true. No specific state information is available.

- `Writing to net`

The server is writing a packet to the network.

6.5.5.3. Delayed-Insert Thread States

These thread states are associated with processing for `DELAYED` inserts (see [Section 12.2.4.2, “INSERT DELAYED Syntax”](#)). Some states are associated with connection threads that process `INSERT DELAYED` statements from clients. Other states are associated with delayed-insert handler threads that insert the rows. There is a delayed-insert handler thread for each table for which `INSERT DELAYED` statements are issued.

States associated with a connection thread that processes an `INSERT DELAYED` statement from the client:

- `allocating local table`

The thread is preparing to feed rows to the delayed-insert handler thread.

- `Creating delayed handler`

The thread is creating a handler for `DELAYED` inserts.

- `got handler lock`

This occurs before the `allocating local table` state and after the `waiting for handler lock` state, when the connection thread gets access to the delayed-insert handler thread.

- `got old table`

This occurs after the `waiting for handler open` state. The delayed-insert handler thread has signaled that it has ended its initialization phase, which includes opening the table for delayed inserts.

- `storing row into queue`

The thread is adding a new row to the list of rows that the delayed-insert handler thread must insert.

- `update`

- `waiting for delay_list`

This occurs during the initialization phase when the thread is trying to find the delayed-insert handler thread for the table, and before attempting to gain access to the list of delayed-insert threads.

- `waiting for handler insert`

An `INSERT DELAYED` handler has processed all pending inserts and is waiting for new ones.

- `waiting for handler lock`

This occurs before the `allocating local table` state when the connection thread waits for access to the delayed-insert handler thread.

- `waiting for handler open`

This occurs after the `Creating delayed handler` state and before the `got old table` state. The delayed-insert handler thread has just been started, and the connection thread is waiting for it to initialize.

States associated with a delayed-insert handler thread that inserts the rows:

- `insert`

The state that occurs just before inserting rows into the table.

- [reschedule](#)

After inserting a number of rows, the delayed-insert thread sleeps to let other threads do work.

- [upgrading lock](#)

A delayed-insert handler is trying to get a lock for the table to insert rows.

- [Waiting for INSERT](#)

A delayed-insert handler is waiting for a connection thread to add rows to the queue (see [storing row into queue](#)).

6.5.5.4. Replication Master Thread States

The following list shows the most common states you may see in the [State](#) column for the master's [Binlog Dump](#) thread. If you see no [Binlog Dump](#) threads on a master server, this means that replication is not running — that is, that no slaves are currently connected.

- [Sending binlog event to slave](#)

Binary logs consist of *events*, where an event is usually an update plus some other information. The thread has read an event from the binary log and is now sending it to the slave.

- [Finished reading one binlog; switching to next binlog](#)

The thread has finished reading a binary log file and is opening the next one to send to the slave.

- [Has sent all binlog to slave; waiting for binlog to be updated](#)

The thread has read all outstanding updates from the binary logs and sent them to the slave. The thread is now idle, waiting for new events to appear in the binary log resulting from new updates occurring on the master.

- [Waiting to finalize termination](#)

A very brief state that occurs as the thread is stopping.

6.5.5.5. Replication Slave I/O Thread States

The following list shows the most common states you see in the [State](#) column for a slave server I/O thread. This state also appears in the [Slave_IO_State](#) column displayed by [SHOW SLAVE STATUS](#), so you can get a good view of what is happening by using that statement.

- [Waiting for master update](#)

The initial state before [Connecting to master](#).

- [Connecting to master](#)

The thread is attempting to connect to the master.

- [Checking master version](#)

A state that occurs very briefly, after the connection to the master is established.

- [Registering slave on master](#)

A state that occurs very briefly after the connection to the master is established.

- [Requesting binlog dump](#)

A state that occurs very briefly, after the connection to the master is established. The thread sends to the master a request for the contents of its binary logs, starting from the requested binary log filename and position.

- **Waiting to reconnect after a failed binlog dump request**

If the binary log dump request failed (due to disconnection), the thread goes into this state while it sleeps, then tries to reconnect periodically. The interval between retries can be specified using the `--master-connect-retry` option.

- **Reconnecting after a failed binlog dump request**

The thread is trying to reconnect to the master.

- **Waiting for master to send event**

The thread has connected to the master and is waiting for binary log events to arrive. This can last for a long time if the master is idle. If the wait lasts for `slave_net_timeout` seconds, a timeout occurs. At that point, the thread considers the connection to be broken and makes an attempt to reconnect.

- **Queueing master event to the relay log**

The thread has read an event and is copying it to the relay log so that the SQL thread can process it.

- **Waiting to reconnect after a failed master event read**

An error occurred while reading (due to disconnection). The thread is sleeping for `master-connect-retry` seconds before attempting to reconnect.

- **Reconnecting after a failed master event read**

The thread is trying to reconnect to the master. When connection is established again, the state becomes **Waiting for master to send event**.

- **Waiting for the slave SQL thread to free enough relay log space**

You are using a non-zero `relay_log_space_limit` value, and the relay logs have grown large enough that their combined size exceeds this value. The I/O thread is waiting until the SQL thread frees enough space by processing relay log contents so that it can delete some relay log files.

- **Waiting for slave mutex on exit**

A state that occurs briefly as the thread is stopping.

6.5.5.6. Replication Slave SQL Thread States

The following list shows the most common states you may see in the `State` column for a slave server SQL thread:

- **Waiting for the next event in relay log**

The initial state before **Reading event from the relay log**.

- **Reading event from the relay log**

The thread has read an event from the relay log so that the event can be processed.

- **Has read all relay log; waiting for the slave I/O thread to update it**

The thread has processed all events in the relay log files, and is now waiting for the I/O thread to write new events to the relay log.

- **Making temp file**

The thread is executing a `LOAD DATA INFILE` statement and is creating a temporary file containing the data from which the slave will read rows.

- `Waiting for slave mutex on exit`

A very brief state that occurs as the thread is stopping.

The `State` column for the I/O thread may also show the text of a statement. This indicates that the thread has read an event from the relay log, extracted the statement from it, and is executing it.

6.5.5.7. Replication Slave Connection Thread States

These thread states occur on a replication slave but are associated with connection threads, not with the I/O or SQL threads.

- `Changing master`

The thread is processing a `CHANGE MASTER` statement.

- `Creating table from master dump`

The slave is creating a table using the `CREATE TABLE` statement contained in the dump from the master. Used for `LOAD TABLE FROM MASTER` and `LOAD DATA FROM MASTER`.

- `Killing slave`

The thread is processing a `SLAVE STOP` statement.

- `Opening master dump table`

This state occurs after `Creating table from master dump`.

- `Reading master dump table data`

This state occurs after `Opening master dump table`.

- `Rebuilding the index on master dump table`

This state occurs after `Reading master dump table data`.

- `starting slave`

The thread is starting the slave threads after processing a successful `LOAD DATA FROM MASTER` load operation.

6.5.5.8. MySQL Cluster Thread States

- `Committing events to binlog`
- `Opening mysql.ndb_apply_status`

- `Processing events`

The thread is processing events for binary logging.

- `Processing events from schema table`

The thread is doing the work of schema replication.

- `Shutting down`

- `Syncing ndb table schema operation and binlog`

This is used to have a correct binary log of schema operations for NDB.

- `Waiting for event from ndbcluster`

The server is acting as an SQL node in a MySQL Cluster, and is connected to a cluster management node.

- [Waiting for first event from ndbcluster](#)
- [Waiting for ndbcluster binlog update to reach current position](#)
- [Waiting for ndbcluster to start](#)
- [Waiting for schema epoch](#)

The thread is waiting for a schema epoch (that is, a global checkpoint).

6.5.5.9. Event Scheduler Thread States

These states occur for the Event Scheduler thread, threads that are created to execute scheduled events, or threads that terminate the scheduler.

- [Clearing](#)

The scheduler thread or a thread that was executing an event is terminating and is about to end.

- [Initialized](#)

The scheduler thread or a thread that will execute an event has been initialized.

- [Waiting for next activation](#)

The scheduler has a non-empty event queue but the next activation is in the future.

- [Waiting for scheduler to stop](#)

The thread issued `SET GLOBAL event_scheduler=OFF` and is waiting for the scheduler to stop.

- [Waiting on empty queue](#)

The scheduler's event queue is empty and it is sleeping.

6.5.6. How Compiling and Linking Affects the Speed of MySQL

Most of the following tests were performed on Linux with the MySQL benchmarks, but they should give some indication for other operating systems and workloads.

You obtain the fastest executables when you link with `-static`.

On Linux, it is best to compile the server with `pgcc` and `-O3`. You need about 200MB memory to compile `sql_yacc.cc` with these options, because `gcc` or `pgcc` needs a great deal of memory to make all functions inline. You should also set `CXX=gcc` when configuring MySQL to avoid inclusion of the `libstdc++` library, which is not needed. Note that with some versions of `pgcc`, the resulting binary runs only on true Pentium processors, even if you use the compiler option indicating that you want the resulting code to work on all x586-type processors (such as AMD).

By using a better compiler and compilation options, you can obtain a 10–30% speed increase in applications. This is particularly important if you compile the MySQL server yourself.

When we tested both the Cygnus CodeFusion and Fujitsu compilers, neither was sufficiently bug-free to allow MySQL to be compiled with optimizations enabled.

The standard MySQL binary distributions are compiled with support for all character sets. When you compile MySQL yourself, you should include support only for the character sets that you are going to use. This is controlled by the `--with-charset` option to `configure`.

Here is a list of some measurements that we have made:

- If you use `pgcc` and compile everything with `-O6`, the `mysqld` server is 1% faster than with `gcc` 2.95.2.
- If you link dynamically (without `-static`), the result is 13% slower on Linux. Note that you still can use a dynamically linked MySQL library for your client applications. It is the server that is most critical for performance.
- For a connection from a client to a server running on the same host, if you connect using TCP/IP rather than a Unix socket file, performance is 7.5% slower. (On Unix, if you connect to the hostname `localhost`, MySQL uses a socket file by default.)
- For TCP/IP connections from a client to a server, connecting to a remote server on another host is 8–11% slower than connecting to a server on the same host, even for connections over 100Mb/s Ethernet.
- When running our benchmark tests using secure connections (all data encrypted with internal SSL support) performance was 55% slower than with unencrypted connections.
- If you compile with `--with-debug=full`, most queries are 20% slower. Some queries may take substantially longer; for example, the MySQL benchmarks run 35% slower. If you use `--with-debug` (without `=full`), the speed decrease is only 15%. For a version of `mysqld` that has been compiled with `--with-debug=full`, you can disable memory checking at runtime by starting it with the `--skip-safemalloc` option. The execution speed should then be close to that obtained when configuring with `--with-debug`.
- On a Sun UltraSPARC-IIe, a server compiled with Forte 5.0 is 4% faster than one compiled with `gcc` 3.2.
- On a Sun UltraSPARC-IIe, a server compiled with Forte 5.0 is 4% faster in 32-bit mode than in 64-bit mode.
- Compiling with `gcc` 2.95.2 for UltraSPARC with the `-mcpu=v8 -wa, -xarch=v8plusa` options gives 4% more performance.
- On Solaris 2.5.1, MIT-pthreads is 8–12% slower than Solaris native threads on a single processor. With greater loads or more CPUs, the difference should be larger.
- Compiling on Linux-x86 using `gcc` without frame pointers (`-fomit-frame-pointer` or `-fomit-frame-pointer -ffixed-ebp`) makes `mysqld` 1–4% faster.

Binary MySQL distributions for Linux that are provided by MySQL AB used to be compiled with `pgcc`. We had to go back to regular `gcc` due to a bug in `pgcc` that would generate binaries that do not run on AMD. We will continue using `gcc` until that bug is resolved. In the meantime, if you have a non-AMD machine, you can build a faster binary by compiling with `pgcc`. The standard MySQL Linux binary is linked statically to make it faster and more portable.

6.5.7. How MySQL Uses Memory

The following list indicates some of the ways that the `mysqld` server uses memory. Where applicable, the name of the system variable relevant to the memory use is given:

- The key buffer is shared by all threads; its size is determined by the `key_buffer_size` variable. Other buffers used by the server are allocated as needed. See [Section 6.5.2, “Tuning Server Parameters”](#).
- Each connection uses some thread-specific space. The following list indicates these and which variables control their size:
 - A stack (default 192KB, variable `thread_stack`)
 - A connection buffer (variable `net_buffer_length`)
 - A result buffer (variable `net_buffer_length`)

The connection buffer and result buffer both begin with a size given by `net_buffer_length` but are dynamically enlarged up to `max_allowed_packet` bytes as needed. The result buffer shrinks to `net_buffer_length` after each SQL statement. While a statement is running, a copy of the current statement string is also allocated.

- All threads share the same base memory.
- When a thread is no longer needed, the memory allocated to it is released and returned to the system unless the thread goes back into the thread cache. In that case, the memory remains allocated.
- Before MySQL 5.1.4, only compressed `MyISAM` tables are memory mapped. As of MySQL 5.1.4, the `myisam_use_mmap` system variable can be set to 1 to enable memory-mapping for all `MyISAM` tables.

- Each request that performs a sequential scan of a table allocates a *read buffer* (variable `read_buffer_size`).
- When reading rows in an arbitrary sequence (for example, following a sort), a *random-read buffer* (variable `read_rnd_buffer_size`) may be allocated in order to avoid disk seeks.
- All joins are executed in a single pass, and most joins can be done without even using a temporary table. Most temporary tables are memory-based hash tables. Temporary tables with a large row length (calculated as the sum of all column lengths) or that contain **BLOB** columns are stored on disk.

If an internal heap table exceeds the size of `tmp_table_size`, MySQL handles this automatically by changing the in-memory heap table to a disk-based **MyISAM** table as necessary. You can also increase the temporary table size by setting the `tmp_table_size` option to `mysqld`, or by setting the SQL option `SQL_BIG_TABLES` in the client program. See [Section 12.5.3, “SET Syntax”](#).

MySQL Enterprise

Subscribers to the MySQL Enterprise Monitor are alerted when temporary tables exceed `tmp_table_size`. Advisors make recommendations for the optimum value of `tmp_table_size` based on actual table usage. For more information about the MySQL Enterprise Monitor please see <http://www.mysql.com/products/enterprise/advisors.html>.

- Most requests that perform a sort allocate a sort buffer and zero to two temporary files depending on the result set size. See [Section B.1.4.4, “Where MySQL Stores Temporary Files”](#).
- Almost all parsing and calculating is done in a local memory store. No memory overhead is needed for small items, so the normal slow memory allocation and freeing is avoided. Memory is allocated only for unexpectedly large strings. This is done with `malloc()` and `free()`.
- For each **MyISAM** table that is opened, the index file is opened once; the data file is opened once for each concurrently running thread. For each concurrent thread, a table structure, column structures for each column, and a buffer of size $3 \times N$ are allocated (where N is the maximum row length, not counting **BLOB** columns). A **BLOB** column requires five to eight bytes plus the length of the **BLOB** data. The **MyISAM** storage engine maintains one extra row buffer for internal use.
- For each table having **BLOB** columns, a buffer is enlarged dynamically to read in larger **BLOB** values. If you scan a table, a buffer as large as the largest **BLOB** value is allocated.
- Handler structures for all in-use tables are saved in a cache and managed as a FIFO. By default, the cache has 64 entries. If a table has been used by two running threads at the same time, the cache contains two entries for the table. See [Section 6.4.8, “How MySQL Opens and Closes Tables”](#).
- A `FLUSH TABLES` statement or `mysqladmin flush-tables` command closes all tables that are not in use at once and marks all in-use tables to be closed when the currently executing thread finishes. This effectively frees most in-use memory. `FLUSH TABLES` does not return until all tables have been closed.

`ps` and other system status programs may report that `mysqld` uses a lot of memory. This may be caused by thread stacks on different memory addresses. For example, the Solaris version of `ps` counts the unused memory between stacks as used memory. You can verify this by checking available swap with `swap -s`. We test `mysqld` with several memory-leakage detectors (both commercial and Open Source), so there should be no memory leaks.

6.5.8. How MySQL Uses Internal Temporary Tables

In some cases, the server creates internal temporary tables while processing queries. A temporary table can be held in memory and processed by the **MEMORY** storage engine, or stored on disk and processed by the **MyISAM** storage engine. Temporary tables can be created under conditions such as these:

- If there is an **ORDER BY** clause and a different **GROUP BY** clause, or if the **ORDER BY** or **GROUP BY** contains columns from tables other than the first table in the join queue, a temporary table is created.
- If you use the `SQL_SMALL_RESULT` option, MySQL uses an in-memory temporary table.
- **DISTINCT** combined with **ORDER BY** may require a temporary table.

You can tell whether a query requires a temporary table by using `EXPLAIN` and checking the `Extra` column to see whether it says

Using `temporary`. See [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#).

Some conditions prevent the use of a `MEMORY` temporary table, in which case the server uses a `MyISAM` table instead:

- Presence of a `TEXT` or `BLOB` column in the table
- Presence of any column in a `GROUP BY` or `DISTINCT` clause larger than 512 bytes

A temporary table that is created initially as a `MEMORY` table might be converted to a `MyISAM` table and stored on disk if it becomes too large. The `max_heap_table_size` system variable determines how large `MEMORY` tables are allowed to grow. It applies to all `MEMORY` tables, including those created with `CREATE TABLE`. However, for internal `MEMORY` tables, the actual maximum size is determined by `max_heap_table_size` in combination with `tmp_table_size`: Whichever value is smaller is the one that applies. If the size of an internal `MEMORY` table exceeds the limit, MySQL automatically converts it to an on-disk `MyISAM` table.

6.5.9. How MySQL Uses DNS

When a new client connects to `mysqld`, `mysqld` spawns a new thread to handle the request. This thread first checks whether the host-name is in the hostname cache. If not, the thread attempts to resolve the hostname:

- If the operating system supports the thread-safe `gethostbyaddr_r()` and `gethostbyname_r()` calls, the thread uses them to perform hostname resolution.
- If the operating system does not support the thread-safe calls, the thread locks a mutex and calls `gethostbyaddr()` and `gethostbyname()` instead. In this case, no other thread can resolve hostnames that are not in the hostname cache until the first thread unlocks the mutex.

You can disable DNS hostname lookups by starting `mysqld` with the `--skip-name-resolve` option. However, in this case, you can use only IP numbers in the MySQL grant tables.

If you have a very slow DNS and many hosts, you can get more performance by either disabling DNS lookups with `--skip-name-resolve` or by increasing the `HOST_CACHE_SIZE` define (default value: 128) and recompiling `mysqld`.

You can disable the hostname cache by starting the server with the `--skip-host-cache` option. To clear the hostname cache, issue a `FLUSH HOSTS` statement or execute the `mysqladmin flush-hosts` command.

To disallow TCP/IP connections entirely, start `mysqld` with the `--skip-networking` option.

6.6. Disk Issues

- Disk seeks are a huge performance bottleneck. This problem becomes more apparent when the amount of data starts to grow so large that effective caching becomes impossible. For large databases where you access data more or less randomly, you can be sure that you need at least one disk seek to read and a couple of disk seeks to write things. To minimize this problem, use disks with low seek times.
- Increase the number of available disk spindles (and thereby reduce the seek overhead) by either symlinking files to different disks or striping the disks:

- Using symbolic links

This means that, for `MyISAM` tables, you symlink the index file and data files from their usual location in the data directory to another disk (that may also be striped). This makes both the seek and read times better, assuming that the disk is not used for other purposes as well. See [Section 6.6.1, “Using Symbolic Links”](#).

- Striping

Striping means that you have many disks and put the first block on the first disk, the second block on the second disk, and the N -th block on the $(N \text{ MOD } \textit{number\_of\_disks})$ disk, and so on. This means if your normal data size is less than the stripe size (or perfectly aligned), you get much better performance. Striping is very dependent on the operating system and the stripe size, so benchmark your application with different stripe sizes. See [Section 6.1.5, “Using Your Own Benchmarks”](#).

The speed difference for striping is very dependent on the parameters. Depending on how you set the striping parameters and

number of disks, you may get differences measured in orders of magnitude. You have to choose to optimize for random or sequential access.

- For reliability, you may want to use RAID 0+1 (striping plus mirroring), but in this case, you need $2 \times N$ drives to hold N drives of data. This is probably the best option if you have the money for it. However, you may also have to invest in some volume-management software to handle it efficiently.
- A good option is to vary the RAID level according to how critical a type of data is. For example, store semi-important data that can be regenerated on a RAID 0 disk, but store really important data such as host information and logs on a RAID 0+1 or RAID N disk. RAID N can be a problem if you have many writes, due to the time required to update the parity bits.
- On Linux, you can get much more performance by using `hdparm` to configure your disk's interface. (Up to 100% under load is not uncommon.) The following `hdparm` options should be quite good for MySQL, and probably for many other applications:

```
hdparm -m 16 -d 1
```

Note that performance and reliability when using this command depend on your hardware, so we strongly suggest that you test your system thoroughly after using `hdparm`. Please consult the `hdparm` manual page for more information. If `hdparm` is not used wisely, filesystem corruption may result, so back up everything before experimenting!

- You can also set the parameters for the filesystem that the database uses:

If you do not need to know when files were last accessed (which is not really useful on a database server), you can mount your filesystems with the `-o noatime` option. That skips updates to the last access time in inodes on the filesystem, which avoids some disk seeks.

On many operating systems, you can set a filesystem to be updated asynchronously by mounting it with the `-o async` option. If your computer is reasonably stable, this should give you more performance without sacrificing too much reliability. (This flag is on by default on Linux.)

6.6.1. Using Symbolic Links

You can move tables and databases from the database directory to other locations and replace them with symbolic links to the new locations. You might want to do this, for example, to move a database to a file system with more free space or increase the speed of your system by spreading your tables to different disk.

The recommended way to do this is simply to symlink databases to a different disk. Symlink tables only as a last resort.

6.6.1.1. Using Symbolic Links for Databases on Unix

On Unix, the way to symlink a database is first to create a directory on some disk where you have free space and then to create a symlink to it from the MySQL data directory.

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test /path/to/datadir
```

MySQL does not support linking one directory to multiple databases. Replacing a database directory with a symbolic link works as long as you do not make a symbolic link between databases. Suppose that you have a database `db1` under the MySQL data directory, and then make a symlink `db2` that points to `db1`:

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

The result is that, or any table `tbl_a` in `db1`, there also appears to be a table `tbl_a` in `db2`. If one client updates `db1.tbl_a` and another client updates `db2.tbl_a`, problems are likely to occur.

However, if you really need to do this, it is possible by altering the source file `mysys/my_symlink.c`, in which you should look for the following statement:

```
if (!(MyFlags & MY_RESOLVE_LINK) ||
    (!lstat(filename, &stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

Change the statement to this:

```
if (1)
```

6.6.1.2. Using Symbolic Links for Tables on Unix

You should not symlink tables on systems that do not have a fully operational `realpath()` call. (Linux and Solaris support `realpath()`). You can check whether your system supports symbolic links by issuing a `SHOW VARIABLES LIKE 'have_symlink'` statement.

Symlinks are fully supported only for **MyISAM** tables. For files used by tables for other storage engines, you may get strange problems if you try to use symbolic links.

The handling of symbolic links for **MyISAM** tables works as follows:

- In the data directory, you always have the table format (`.frm`) file, the data (`.MYD`) file, and the index (`.MYI`) file. The data file and index file can be moved elsewhere and replaced in the data directory by symlinks. The format file cannot.
- You can symlink the data file and the index file independently to different directories.
- You can instruct a running MySQL server to perform the symlinking by using the `DATA DIRECTORY` and `INDEX DIRECTORY` options to `CREATE TABLE`. See [Section 12.1.10, “CREATE TABLE Syntax”](#). Alternatively, symlinking can be accomplished manually from the command line using `ln -s` if `mysqld` is not running.
- `myisamchk` does not replace a symlink with the data file or index file. It works directly on the file to which the symlink points. Any temporary files are created in the directory where the data file or index file is located. The same is true for the `ALTER TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.

Note

When you drop a table that is using symlinks, *both the symlink and the file to which the symlink points are dropped*. This is an extremely good reason why you should *not* run `mysqld` as the system `root` or allow system users to have write access to MySQL database directories.

- If you rename a table with `ALTER TABLE ... RENAME` and you do not move the table to another database, the symlinks in the database directory are renamed to the new names and the data file and index file are renamed accordingly.
- If you use `ALTER TABLE ... RENAME` to move a table to another database, the table is moved to the other database directory. The old symlinks and the files to which they pointed are deleted. In other words, the new table is not symlinked.
- If you are not using symlinks, you should use the `--skip-symbolic-links` option to `mysqld` to ensure that no one can use `mysqld` to drop or rename a file outside of the data directory.

Table symlink operations that are not yet supported:

- `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.
- `BACKUP TABLE` and `RESTORE TABLE` do not respect symbolic links.
- The `.frm` file must *never* be a symbolic link (as indicated previously, only the data and index files can be symbolic links). Attempting to do this (for example, to make synonyms) produces incorrect results. Suppose that you have a database `db1` under the MySQL data directory, a table `tbl1` in this database, and in the `db1` directory you make a symlink `tbl2` that points to `tbl1`:

```
shell> cd /path/to/datadir/db1
shell> ln -s tbl1.frm tbl2.frm
shell> ln -s tbl1.MYD tbl2.MYD
shell> ln -s tbl1.MYI tbl2.MYI
```

Problems result if one thread reads `db1.tbl1` and another thread updates `db1.tbl2`:

- The query cache is “fooled” (it has no way of knowing that `tbl1` has not been updated, so it returns outdated results).

- `ALTER` statements on `tbl2` fail.

6.6.1.3. Using Symbolic Links for Databases on Windows

Symbolic links are enabled by default for all Windows servers. This enables you to put a database directory on a different disk by setting up a symbolic link to it. This is similar to the way that database symbolic links work on Unix, although the procedure for setting up the link is different. If you do not need symbolic links, you can disable them using the `--skip-symbolic-links` option.

On Windows, create a symbolic link to a MySQL database by creating a file in the data directory that contains the path to the destination directory. The file should be named `db_name.sym`, where `db_name` is the database name.

Suppose that the MySQL data directory is `C:\mysql\data` and you want to have database `foo` located at `D:\data\foo`. Set up a symlink using this procedure

1. Make sure that the `D:\data\foo` directory exists by creating it if necessary. If you already have a database directory named `foo` in the data directory, you should move it to `D:\data`. Otherwise, the symbolic link will be ineffective. To avoid problems, make sure that the server is not running when you move the database directory.
2. Create a text file `C:\mysql\data\foo.sym` that contains the pathname `D:\data\foo\`.

After this, all tables created in the database `foo` are created in `D:\data\foo`.

The following limitations apply to the use of `.sym` files for database symbolic linking on Windows:

- The symbolic link is not used if a directory with the same name as the database exists in the MySQL data directory.
- The `--innodb_file_per_table` option cannot be used.

Chapter 7. Client and Utility Programs

There are many different MySQL client programs that connect to the server to access databases or perform administrative tasks. Other utilities are available as well. These do not establish a client connection with the server but perform MySQL-related operations.

This chapter provides a brief overview of these programs and then a more detailed description of each one. Each program's description indicates its invocation syntax and the options that it understands. See [Chapter 4, *Using MySQL Programs*](#), for general information on invoking programs and specifying program options.

7.1. Overview of Client and Utility Programs

The following list briefly describes the MySQL client programs and utilities:

- **innochecksum**
An offline InnoDB offline file checksum utility. See [Section 7.2, “innochecksum — Offline InnoDB File Checksum Utility”](#).
- **my_print_defaults**
A utility that are present in option groups of option files. See [Section 7.3, “my_print_defaults — Display Options from Option Files”](#).
- **myisam_ftdump**
A utility that displays information about full-text indexes in MyISAM tables. See [Section 7.4, “myisam_ftdump — Display Full-Text Index information”](#).
- **myisamchk**
A utility to describe, check, optimize, and repair MyISAM tables. See [Section 7.5, “myisamchk — MyISAM Table-Maintenance Utility”](#).
- **myisamlog, isamlog**
A utility that processes the contents of a MyISAM log file. See [Section 7.6, “myisamlog — Display MyISAM Log File Contents”](#).
- **myisampack**
A utility that compresses MyISAM tables to produce smaller read-only tables. See [Section 7.7, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#).
- **mysql**
The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See [Section 7.8, “mysql — The MySQL Command-Line Tool”](#).
- **mysqlaccess**
A script that checks the access privileges for a hostname, username, and database combination. See [Section 7.9, “mysqlaccess — Client for Checking Access Privileges”](#).
- **mysqladmin**
A client that performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. **mysqladmin** can also be used to retrieve version, process, and status information from the server. See [Section 7.10, “mysqladmin — Client for Administering a MySQL Server”](#).
- **mysqlbinlog**
A utility for reading statements from a binary log. The log of executed statements contained in the binary log files can be used to help recover from a crash. See [Section 7.11, “mysqlbinlog — Utility for Processing Binary Log Files”](#).
- **mysqlcheck**

A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See [Section 7.12, “mysqlcheck — A Table Maintenance and Repair Program”](#).

- `mysqldump`

A client that dumps a MySQL database into a file as SQL, text, or XML. See [Section 7.13, “mysqldump — A Database Backup Program”](#).

- `mysqlhotcopy`

A utility that quickly makes backups of **MyISAM** tables while the server is running. See [Section 7.14, “mysqlhotcopy — A Database Backup Program”](#).

- `mysqlimport`

A client that imports text files into their respective tables using `LOAD DATA INFILE`. See [Section 7.15, “mysqlimport — A Data Import Program”](#).

- `mysqlshow`

A client that displays information about databases, tables, columns, and indexes. See [Section 7.16, “mysqlshow — Display Database, Table, and Column Information”](#).

- `mysqlslap`

A client that is designed to emulate client load for a MySQL server and report the timing of each stage. It works as if multiple clients are accessing the server. See [Section 7.17, “mysqlslap — Load Emulation Client”](#).

- `mysql_convert_table_format`

A utility that converts tables in a database to use a given storage engine. See [Section 7.18, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”](#).

- `mysql_find_rows`

A utility that reads files containing SQL statements (such as update logs) and extracts statements that match a given regular expression. See [Section 7.19, “mysql_find_rows — Extract SQL Statements from Files”](#).

- `mysql_fix_extensions`

A utility that converts the extensions for **MyISAM** table files to lowercase. This can be useful after transferring the files from a system with case-insensitive filenames to a system with case-sensitive filenames. See [Section 7.20, “mysql_fix_extensions — Make Table Filename Extensions Lowercase”](#).

- `mysql_setpermission`

A utility for interactively setting permissions in the MySQL grant tables. See [Section 7.21, “mysql_setpermission — Interactively Set Permissions in Grant Tables”](#).

- `mysql_tableinfo`

A utility that generates database metadata. [Section 7.22, “mysql_tableinfo — Generate Database Metadata”](#).

- `mysql_waitpid`

A utility that kills the process with a given process ID. See [Section 7.23, “mysql_waitpid — Kill Process and Wait for Its Termination”](#).

- `mysql_zap`

A utility that kills processes that match a pattern. See [Section 7.24, “mysql_zap — Kill Processes That Match a Pattern”](#).

- `perror`

A utility that displays the meaning of system or MySQL error codes. See [Section 7.25, “perror — Explain Error Codes”](#).

- `replace`

A utility program that performs string replacement in the input text. See [Section 7.26, “replace — A String-Replacement Utility”](#).

- `resolveip`

A utility program that resolves a hostname to an IP address or vice versa. See [Section 7.27, “resolveip — Resolve Hostname to IP Address or Vice Versa”](#).

- `resolve_stack_dump`

A utility program that resolves a numeric stack trace dump to symbols. See [Section 7.28, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”](#).

MySQL AB also provides a number of GUI tools for administering and otherwise working with MySQL servers. For basic information about these, see [Chapter 4, Using MySQL Programs](#).

Each MySQL program takes many different options. Most programs provide a `--help` option that you can use to get a full description of the program's different options. For example, try `mysql --help`.

MySQL client programs that communicate with the server using the MySQL client/server library use the following environment variables:

<code>MYSQL_UNIX_PORT</code>	The default Unix socket file; used for connections to <code>localhost</code>
<code>MYSQL_TCP_PORT</code>	The default port number; used for TCP/IP connections
<code>MYSQL_PWD</code>	The default password
<code>MYSQL_DEBUG</code>	Debug trace options when debugging
<code>TMPDIR</code>	The directory where temporary tables and files are created

Use of `MYSQL_PWD` is insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

You can override the default option values or values specified in environment variables for all standard programs by specifying options in an option file or on the command line. See [Section 4.3, “Specifying Program Options”](#).

7.2. `innochecksum` — Offline InnoDB File Checksum Utility

`innochecksum` prints checksums for InnoDB files.

Invoke `innochecksum` like this:

```
shell> innochecksum [options] file_name
```

`innochecksum` understands the options described in the following list. For options that refer to page numbers, the numbers are zero-based.

- `-c`

Print a count of the number of pages in the file.

- `-d`

Debug mode; prints checksums for each page.

- `-e num`

End at this page number.

- `-p num`

Check only this page number.

- `-s num`

Start at this page number.

- `-v`

Verbose mode; print a progress indicator every five seconds.

7.3. `my_print_defaults` — Display Options from Option Files

`my_print_defaults` displays the options that are present in option groups of option files. The output indicates what options will be used by programs that read the specified option groups. For example, the `mysqlcheck` program reads the `[mysqlcheck]` and `[client]` option groups. To see what options are present in those groups in the standard option files, invoke `my_print_defaults` like this:

```
shell> my_print_defaults mysqlcheck client
--user=myusername
--password=secret
--host=localhost
```

The output consists of options, one per line, in the form that they would be specified on the command line.

`my_print_defaults` understands the following options:

- `--help, -?`

Display a help message and exit.

- `--config-file=file_name, --defaults-file=file_name, -c file_name`

Read only the given option file.

- `--debug=debug_options, -# debug_options`

Write a debugging log. The `debug_options` string often is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/my_print_defaults.trace'`.

- `--defaults-extra-file=file_name, --extra-file=file_name, -e file_name`

Read this option file after the global option file but (on Unix) before the user option file.

- `--defaults-group-suffix=suffix, -g suffix`

In addition to the groups named on the command line, read groups that have the given suffix.

- `--no-defaults, -n`

Return an empty string.

- `--verbose, -v`

Verbose mode. Print more information about what the program does.

- `--version, -V`

Display version information and exit.

7.4. `myisam_ftdump` — Display Full-Text Index information

`myisam_ftdump` displays information about `FULLTEXT` indexes in `MyISAM` tables. It reads the `MyISAM` index file directly, so it

must be run on the server host where the table is located

Invoke `myisam_ftdump` like this:

```
shell> myisam_ftdump [options] tbl_name index_num
```

The `tbl_name` argument should be the name of a **MyISAM** table. You can also specify a table by naming its index file (the file with the `.MYI` suffix). If you do not invoke `myisam_ftdump` in the directory where the table files are located, the table or index file name must be preceded by the pathname to the table's database directory. Index numbers begin with 0.

Example: Suppose that the `test` database contains a table named `mytexttable1` that has the following definition:

```
CREATE TABLE mytexttable
(
  id   INT NOT NULL,
  txt  TEXT NOT NULL,
  PRIMARY KEY (id),
  FULLTEXT (txt)
);
```

The index on `id` is index 0 and the `FULLTEXT` index on `txt` is index 1. If your working directory is the `test` database directory, invoke `myisam_ftdump` as follows:

```
shell> myisam_ftdump mytexttable 1
```

If the pathname to the `test` database directory is `/usr/local/mysql/data/test`, you can also specify the table name argument using that pathname. This is useful if you do not invoke `myisam_ftdump` in the database directory:

```
shell> myisam_ftdump /usr/local/mysql/data/test/mytexttable 1
```

`myisam_ftdump` understands the following options:

- `--help, -h -?`
Display a help message and exit.
- `--count, -c`
Calculate per-word statistics (counts and global weights).
- `--dump, -d`
Dump the index, including data offsets and word weights.
- `--length, -l`
Report the length distribution.
- `--stats, -s`
Report global index statistics. This is the default operation if no other operation is specified.
- `--verbose, -v`
Verbose mode. Print more output about what the program does.

7.5. `myisamchk` — MyISAM Table-Maintenance Utility

The `myisamchk` utility gets information about your database tables or checks, repairs, or optimizes them. `myisamchk` works with **MyISAM** tables (tables that have `.MYD` and `.MYI` files for storing data and indexes).

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation

might cause data loss. Possible causes include but are not limited to filesystem errors.

Invoke `myisamchk` like this:

```
shell> myisamchk [options] tbl_name ...
```

The *options* specify what you want `myisamchk` to do. They are described in the following sections. You can also get a list of options by invoking `myisamchk --help`.

With no options, `myisamchk` simply checks your table as the default operation. To get more information or to tell `myisamchk` to take corrective action, specify options as described in the following discussion.

tbl_name is the database table you want to check or repair. If you run `myisamchk` somewhere other than in the database directory, you must specify the path to the database directory, because `myisamchk` has no idea where the database is located. In fact, `myisamchk` doesn't actually care whether the files you are working on are located in a database directory. You can copy the files that correspond to a database table into some other location and perform recovery operations on them there.

You can name several tables on the `myisamchk` command line if you wish. You can also specify a table by naming its index file (the file with the `.MYI` suffix). This allows you to specify all tables in a directory by using the pattern `*.MYI`. For example, if you are in a database directory, you can check all the `MyISAM` tables in that directory like this:

```
shell> myisamchk *.MYI
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell> myisamchk /path/to/database_dir/*.MYI
```

You can even check all tables in all databases by specifying a wildcard with the path to the MySQL data directory:

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

The recommended way to quickly check all `MyISAM` tables is:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
```

If you want to check all `MyISAM` tables and repair any that are corrupted, you can use the following command:

```
shell> myisamchk --silent --force --fast --update-state \  
--key_buffer_size=64M --sort_buffer_size=64M \  
--read_buffer_size=1M --write_buffer_size=1M \  
/path/to/datadir/*/*.MYI
```

This command assumes that you have more than 64MB free. For more information about memory allocation with `myisamchk`, see [Section 7.5.5, “myisamchk Memory Usage”](#).

MySQL Enterprise

For expert advice on checking and repairing tables, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

Important

You must ensure that no other program is using the tables while you are running `myisamchk`. The most effective means of doing so is to shut down the MySQL server while running `myisamchk`, or to lock all tables that `myisamchk` is being used on.

Otherwise, when you run `myisamchk`, it may display the following error message:

```
warning: clients are using or haven't closed the table properly
```

This means that you are trying to check a table that has been updated by another program (such as the `mysqld` server) that hasn't yet closed the file or that has died without closing the file properly, which can sometimes lead to the corruption of one or more `MyISAM` tables.

If `mysqld` is running, you must force it to flush any table modifications that are still buffered in memory by using `FLUSH TABLES`. You should then ensure that no one is using the tables while you are running `myisamchk`.

However, the easiest way to avoid this problem is to use `CHECK TABLE` instead of `myisamchk` to check tables. See [Section 12.5.2.3, “CHECK TABLE Syntax”](#).

7.5.1. `myisamchk` General Options

The options described in this section can be used for any type of table maintenance operation performed by `myisamchk`. The sections following this one describe options that pertain only to specific operations, such as table checking or repairing.

- `--help, -?`

Display a help message and exit.

- `--debug=debug_options, -# debug_options`

Write a debugging log. The `debug_options` string often is `'d:t:o,file_name'`.

- `--silent, -s`

Silent mode. Write output only when errors occur. You can use `-s` twice (`-ss`) to make `myisamchk` very silent.

- `--verbose, -v`

Verbose mode. Print more information about what the program does. This can be used with `-d` and `-e`. Use `-v` multiple times (`-vv`, `-vvv`) for even more output.

- `--version, -V`

Display version information and exit.

- `--wait, -w`

Instead of terminating with an error if the table is locked, wait until the table is unlocked before continuing. If you are running `mysqld` with external locking disabled, the table can be locked only by another `myisamchk` command.

You can also set the following variables by using `--var_name=value` syntax:

Variable	Default Value
<code>decode_bits</code>	9
<code>ft_max_word_len</code>	version-dependent
<code>ft_min_word_len</code>	4
<code>ft_stopword_file</code>	built-in list
<code>key_buffer_size</code>	523264
<code>myisam_block_size</code>	1024
<code>read_buffer_size</code>	262136
<code>sort_buffer_size</code>	2097144
<code>sort_key_blocks</code>	16
<code>stats_method</code>	nulls_unequal
<code>write_buffer_size</code>	262136

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. However, this syntax is deprecated as of MySQL 4.0.

The possible `myisamchk` variables and their default values can be examined with `myisamchk --help`:

`sort_buffer_size` is used when the keys are repaired by sorting keys, which is the normal case when you use `--recover`.

`key_buffer_size` is used when you are checking the table with `--extend-check` or when the keys are repaired by inserting keys row by row into the table (like when doing normal inserts). Repairing through the key buffer is used in the following cases:

- You use `--safe-recover`.
- The temporary files needed to sort the keys would be more than twice as big as when creating the key file directly. This is often the case when you have large key values for `CHAR`, `VARCHAR`, or `TEXT` columns, because the sort operation needs to store the complete key values as it proceeds. If you have lots of temporary space and you can force `myisamchk` to repair by sorting, you can use the `--sort-recover` option.

Repairing through the key buffer takes much less disk space than using sorting, but is also much slower.

If you want a faster repair, set the `key_buffer_size` and `sort_buffer_size` variables to about 25% of your available memory. You can set both variables to large values, because only one of them is used at a time.

`myisam_block_size` is the size used for index blocks.

`stats_method` influences how `NULL` values are treated for index statistics collection when the `--analyze` option is given. It acts like the `myisam_stats_method` system variable. For more information, see the description of `myisam_stats_method` in [Section 5.2.3, “System Variables”](#), and [Section 6.4.7, “MyISAM Index Statistics Collection”](#). For MySQL 5.1, `stats_method` was added in MySQL 5.0.14. For older versions, the statistics collection method is equivalent to `nulls_equal`.

The `ft_min_word_len` and `ft_max_word_len` variables are available as of MySQL 4.0.0. `ft_stopword_file` is available as of MySQL 4.0.19.

`ft_min_word_len` and `ft_max_word_len` indicate the minimum and maximum word length for `FULLTEXT` indexes. `ft_stopword_file` names the stopword file. These need to be set under the following circumstances.

If you use `myisamchk` to perform an operation that modifies table indexes (such as repair or analyze), the `FULLTEXT` indexes are rebuilt using the default full-text parameter values for minimum and maximum word length and the stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in `MyISAM` index files. To avoid the problem if you have modified the minimum or maximum word length or the stopword file in the server, specify the same `ft_min_word_len`, `ft_max_word_len`, and `ft_stopword_file` values to `myisamchk` that you use for `mysqld`. For example, if you have set the minimum word length to 3, you can repair a table with `myisamchk` like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that `myisamchk` and the server use the same values for full-text parameters, you can place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using `myisamchk` is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE`. These statements are performed by the server, which knows the proper full-text parameter values to use.

7.5.2. `myisamchk` Check Options

`myisamchk` supports the following options for table checking operations:

- `--check, -c`

Check the table for errors. This is the default operation if you specify no option that selects an operation type explicitly.

- `--check-only-changed, -C`

Check only tables that have changed since the last check.

- `--extend-check, -e`

Check the table very thoroughly. This is quite slow if the table has many indexes. This option should only be used in extreme cases. Normally, `myisamchk` or `myisamchk --medium-check` should be able to determine whether there are any errors in the table.

If you are using `--extend-check` and have plenty of memory, setting the `key_buffer_size` variable to a large value helps the repair operation run faster.

- `--fast, -F`

Check only tables that haven't been closed properly.

- `--force, -f`

Do a repair operation automatically if `myisamchk` finds any errors in the table. The repair type is the same as that specified with the `--recover` or `-r` option.

- `--information, -i`

Print informational statistics about the table that is checked.

- `--medium-check, -m`

Do a check that is faster than an `--extend-check` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--read-only, -T`

Don't mark the table as checked. This is useful if you use `myisamchk` to check a table that is in use by some other application that doesn't use locking, such as `mysqld` when run with external locking disabled.

- `--update-state, -U`

Store information in the `.MYI` file to indicate when the table was checked and whether the table crashed. This should be used to get full benefit of the `--check-only-changed` option, but you shouldn't use this option if the `mysqld` server is using the table and you are running it with external locking disabled.

7.5.3. `myisamchk` Repair Options

`myisamchk` supports the following options for table repair operations:

- `--backup, -B`

Make a backup of the `.MYD` file as `file_name-time.BAK`

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--correct-checksum`

Correct the checksum information for the table.

- `--data-file-length=len, -D len`

Maximum length of the data file (when re-creating data file when it is “full”).

- `--extend-check, -e`

Do a repair that tries to recover every possible row from the data file. Normally, this also finds a lot of garbage rows. Don't use this

option unless you are desperate.

- `--force, -f`

Overwrite old intermediate files (files with names like `tbl_name.TMD`) instead of aborting.

- `--keys-used=val, -k val`

For `myisamchk`, the option value is a bit-value that indicates which indexes to update. Each binary bit of the option value corresponds to a table index, where the first index is bit 0. An option value of 0 disables updates to all indexes, which can be used to get faster inserts. Deactivated indexes can be reactivated by using `myisamchk -r`.

- `--no-symlinks, -l`

Do not follow symbolic links. Normally `myisamchk` repairs the table that a symlink points to. This option does not exist as of MySQL 4.0 because versions from 4.0 on do not remove symlinks during repair operations.

- `--max-record-length=len`

Skip rows larger than the given length if `myisamchk` cannot allocate memory to hold them.

- `--parallel-recover, -p`

Uses the same technique as `-r` and `-n`, but creates all the keys in parallel, using different threads. *This is beta-quality code. Use at your own risk!*

- `--quick, -q`

Achieve a faster repair by not modifying the data file. You can specify this option twice to force `myisamchk` to modify the original data file in case of duplicate keys.

- `--recover, -r`

Do a repair that can fix almost any problem except unique keys that aren't unique (which is an extremely unlikely error with `MyISAM` tables). If you want to recover a table, this is the option to try first. You should try `--safe-recover` only if `myisamchk` reports that the table can't be recovered using `--recover`. (In the unlikely case that `--recover` fails, the data file remains intact.)

If you have lots of memory, you should increase the value of `sort_buffer_size`.

- `--safe-recover, -o`

Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found. This is an order of magnitude slower than `--recover`, but can handle a couple of very unlikely cases that `--recover` cannot. This recovery method also uses much less disk space than `--recover`. Normally, you should repair first with `--recover`, and then with `--safe-recover` only if `--recover` fails.

If you have lots of memory, you should increase the value of `key_buffer_size`.

- `--set-character-set=name`

Change the character set used by the table indexes. This option was replaced by `--set-collation` in MySQL 5.0.3.

- `--set-collation=name`

Specify the collation to use for sorting table indexes. The character set name is implied by the first part of the collation name.

- `--sort-recover, -n`

Force `myisamchk` to use sorting to resolve the keys even if the temporary files would be very large.

- `--tmpdir=path, -t path`

Path of the directory to be used for storing temporary files. If this is not set, `myisamchk` uses the value of the `TMPDIR` environment variable. `tmpdir` can be set to a list of directory paths that are used successively in round-robin fashion for creating temporary files. The separator character between directory names is the colon (':') on Unix and the semicolon (';') on Windows, Net-

Ware, and OS/2.

- `--unpack, -u`

Unpack a table that was packed with `myisampack`.

7.5.4. Other `myisamchk` Options

`myisamchk` supports the following options for actions other than table checks and repairs:

- `--analyze, -a`

Analyze the distribution of key values. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use. To obtain information about the key distribution, use a `myisamchk --description --verbose tbl_name` command or the `SHOW INDEX FROM tbl_name` statement.

MySQL Enterprise

For expert advice on optimizing tables, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `--block-search=offset, -b offset`

Find the record that a block at the given offset belongs to.

- `--description, -d`

Print some descriptive information about the table.

- `--set-auto-increment[=value], -A[value]`

Force `AUTO_INCREMENT` numbering for new records to start at the given value (or higher, if there are existing records with `AUTO_INCREMENT` values this large). If `value` is not specified, `AUTO_INCREMENT` numbers for new records begin with the largest value currently in the table, plus one.

- `--sort-index, -S`

Sort the index tree blocks in high-low order. This optimizes seeks and makes table scans that use indexes faster.

- `--sort-records=N, -R N`

Sort records according to a particular index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index. (The first time you use this option to sort a table, it may be very slow.) To determine a table's index numbers, use `SHOW INDEX`, which displays a table's indexes in the same order that `myisamchk` sees them. Indexes are numbered beginning with 1.

If keys are not packed (`PACK_KEYS=0`), they have the same length, so when `myisamchk` sorts and moves records, it just overwrites record offsets in the index. If keys are packed (`PACK_KEYS=1`), `myisamchk` must unpack key blocks first, then re-create indexes and pack the key blocks again. (In this case, re-creating indexes is faster than updating offsets for each index.)

7.5.5. `myisamchk` Memory Usage

Memory allocation is important when you run `myisamchk`. `myisamchk` uses no more memory than its memory-related variables are set to. If you are going to use `myisamchk` on very large tables, you should first decide how much memory you want it to use. The default is to use only about 3MB to perform repairs. By using larger values, you can get `myisamchk` to operate faster. For example, if you have more than 32MB RAM, you could use options such as these (in addition to any other options you might specify):

```
shell> myisamchk --sort_buffer_size=16M --key_buffer_size=16M \
--read_buffer_size=1M --write_buffer_size=1M ...
```

Using `--sort_buffer_size=16M` should probably be enough for most cases.

Be aware that `myisamchk` uses temporary files in `TMPDIR`. If `TMPDIR` points to a memory filesystem, you may easily get out of

memory errors. If this happens, run `myisamchk` with the `--tmpdir=path` option to specify some directory located on a filesystem that has more space.

When repairing, `myisamchk` also needs a lot of disk space:

- Double the size of the data file (the original file and a copy). This space is not needed if you do a repair with `--quick`; in this case, only the index file is re-created. This space is needed on the same filesystem as the original data file! (The copy is created in the same directory as the original.)
- Space for the new index file that replaces the old one. The old index file is truncated at the start of the repair operation, so you usually ignore this space. This space is needed on the same filesystem as the original index file!
- When using `--recover` or `--sort-recover` (but not when using `--safe-recover`), you need space for a sort buffer. The following formula yields the amount of space required:

```
(largest_key + row_pointer_length) × number_of_rows × 2
```

You can check the length of the keys and the `row_pointer_length` with `myisamchk -dv tbl_name`. This space is allocated in the temporary directory (specified by `TMPDIR` or `--tmpdir=path`).

If you have a problem with disk space during repair, you can try `--safe-recover` instead of `--recover`.

7.6. `myisamlog` — Display MyISAM Log File Contents

`myisamlog` processes the contents of a `MyISAM` log file.

Invoke `myisamlog` like this:

```
shell> myisamlog [options] [log_file [tbl_name] ...]
shell> isamlog [options] [log_file [tbl_name] ...]
```

The default operation is update (`-u`). If a recovery is done (`-r`), all writes and possibly updates and deletes are done and errors are only counted. The default log file name is `myisam.log` for `myisamlog` and `isam.log` for `isamlog` if no `log_file` argument is given. If tables are named on the command line, only those tables are updated.

`myisamlog` understands the following options:

- `-?`, `-I`
Display a help message and exit.
- `-c N`
Execute only `N` commands.
- `-f N`
Specify the maximum number of open files.
- `-i`
Display extra information before exiting.
- `-o offset`
Specify the starting offset.
- `-p N`
Remove `N` components from path.
- `-r`

Perform a recovery operation.

- `-R record_pos_file record_pos`

Specify record position file and record position.

- `-u`

Perform an update operation.

- `-v`

Verbose mode. Print more output about what the program does. This option can be given multiple times to produce more and more output.

- `-w write_file`

Specify the write file.

- `-V`

Display version information.

7.7. `myisampack` — Generate Compressed, Read-Only MyISAM Tables

The `myisampack` utility compresses [MyISAM](#) tables. `myisampack` works by compressing each column in the table separately. Usually, `myisampack` packs the data file 40%-70%.

When the table is used later, the server reads into memory the information needed to decompress columns. This results in much better performance when accessing individual rows, because you only have to uncompress exactly one row.

MySQL uses `mmap()` when possible to perform memory mapping on compressed tables. If `mmap()` does not work, MySQL falls back to normal read/write file operations.

Please note the following:

- If the `mysqld` server was invoked with external locking disabled, it is not a good idea to invoke `myisampack` if the table might be updated by the server during the packing process. It is safest to compress tables with the server stopped.
- After packing a table, it becomes read-only. This is generally intended (such as when accessing packed tables on a CD). Allowing writes to a packed table is on our TODO list, but with low priority.

Invoke `myisampack` like this:

```
shell> myisampack [options] file_name ...
```

Each filename argument should be the name of an index (`.MYI`) file. If you are not in the database directory, you should specify the pathname to the file. It is permissible to omit the `.MYI` extension.

After you compress a table with `myisampack`, you should use `myisamchk -rq` to rebuild its indexes. [Section 7.5, “myisamchk — MyISAM Table-Maintenance Utility”](#).

`myisampack` supports the following options:

- `--help, -?`

Display a help message and exit.

- `--backup, -b`

Make a backup of each table's data file using the name `tbl_name.OLD`.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.10.1](#), “The Character Set Used for Data and Sorting”.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The `debug_options` string often is `'d:t:o,file_name'`.

- `--force, -f`

Produce a packed table even if it becomes larger than the original or if the intermediate file from an earlier invocation of `myisam-pack` exists. (`myisam-pack` creates an intermediate file named `tbl_name.TMD` in the database directory while it compresses the table. If you kill `myisam-pack`, the `.TMD` file might not be deleted.) Normally, `myisam-pack` exits with an error if it finds that `tbl_name.TMD` exists. With `--force`, `myisam-pack` packs the table anyway.

- `--join=big_tbl_name, -j big_tbl_name`

Join all tables named on the command line into a single table `big_tbl_name`. All tables that are to be combined *must* have identical structure (same column names and types, same indexes, and so forth).

- `--silent, -s`

Silent mode. Write output only when errors occur.

- `--test, -t`

Do not actually pack the table, just test packing it.

- `--tmpdir=path, -T path`

Use the named directory as the location where `myisam-pack` creates temporary files.

- `--verbose, -v`

Verbose mode. Write information about the progress of the packing operation and its result.

- `--version, -V`

Display version information and exit.

- `--wait, -w`

Wait and retry if the table is in use. If the `mysql` server was invoked with external locking disabled, it is not a good idea to invoke `myisam-pack` if the table might be updated by the server during the packing process.

The following sequence of commands illustrates a typical table compression session:

```
shell> ls -l station.*
-rw-rw-r-- 1 monty my 994128 Apr 17 19:00 station.MYD
-rw-rw-r-- 1 monty my 53248 Apr 17 19:00 station.MYI
-rw-rw-r-- 1 monty my 5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file: station
Isam-version: 2
Creation time: 1996-03-13 10:08:58
Recover time: 1997-02-02 3:06:43
Data records: 1192 Deleted blocks: 0
Datafile parts: 1192 Deleted data: 0
Datafile pointer (bytes): 2 Keyfile pointer (bytes): 2
Max datafile length: 54657023 Max keyfile length: 33554431
Recordlength: 834
Record format: Fixed length

table description:
Key Start Len Index Type Root Blocksize Rec/key
1 2 4 unique unsigned long 1024 1024 1
```

```

2  32  30  multip. text          10240      1024      1

Field Start Length Type
1      1      1
2      2      4
3      6      4
4     10      1
5     11     20
6     31      1
7     32     30
8     62     35
9     97     35
10    132     35
11    167      4
12    171     16
13    187     35
14    222      4
15    226     16
16    242     20
17    262     20
18    282     20
19    302     30
20    332      4
21    336      4
22    340      1
23    341      8
24    349      8
25    357      8
26    365      2
27    367      2
28    369      4
29    373      4
30    377      1
31    378      2
32    380      8
33    388      4
34    392      4
35    396      4
36    400      4
37    404      1
38    405      4
39    409      4
40    413      4
41    417      4
42    421      4
43    425      4
44    429     20
45    449     30
46    479      1
47    480      1
48    481     79
49    560     79
50    639     79
51    718     79
52    797      8
53    805      1
54    806      1
55    807     20
56    827      4
57    831      4

shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

normal:      20  empty-space:   16  empty-zero:    12  empty-fill:   11
pre-space:    0  end-space:    12  table-lookups:   5  zero:         7
Original trees: 57  After join: 17
- Compressing file
87.14%
Remember to run myisamchk -rq on compressed tables

shell> ls -l station.*
-rw-rw-r--  1 monty  my          127874 Apr 17 19:00 station.MYD
-rw-rw-r--  1 monty  my          55296 Apr 17 19:04 station.MYI
-rw-rw-r--  1 monty  my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file:      station
Isam-version:     2
Creation time:    1996-03-13 10:08:58
Recover time:     1997-04-17 19:04:26
Data records:     1192  Deleted blocks:          0
Datafile parts:   1192  Deleted data:          0
Datafile pointer (bytes): 3  Keyfile pointer (bytes): 1
Max datafile length: 16777215  Max keyfile length: 131071
Recordlength:     834
Record format:    Compressed

table description:

```

Key	Start	Len	Index	Type	Root	Blocksize	Rec/key
1	2	4	unique	unsigned long	10240	1024	1
2	32	30	multip.	text	54272	1024	1

Field	Start	Length	Type	Huff tree	Bits
1	1	1	constant	1	0
2	2	4	zerofill(1)	2	9
3	6	4	no zeros, zerofill(1)	2	9
4	10	1		3	9
5	11	20	table-lookup	4	0
6	31	1		3	9
7	32	30	no endspace, not_always	5	9
8	62	35	no endspace, not_always, no empty	6	9
9	97	35	no empty	7	9
10	132	35	no endspace, not_always, no empty	6	9
11	167	4	zerofill(1)	2	9
12	171	16	no endspace, not_always, no empty	5	9
13	187	35	no endspace, not_always, no empty	6	9
14	222	4	zerofill(1)	2	9
15	226	16	no endspace, not_always, no empty	5	9
16	242	20	no endspace, not_always	8	9
17	262	20	no endspace, no empty	8	9
18	282	20	no endspace, no empty	5	9
19	302	30	no endspace, no empty	6	9
20	332	4	always zero	2	9
21	336	4	always zero	2	9
22	340	1		3	9
23	341	8	table-lookup	9	0
24	349	8	table-lookup	10	0
25	357	8	always zero	2	9
26	365	2		2	9
27	367	2	no zeros, zerofill(1)	2	9
28	369	4	no zeros, zerofill(1)	2	9
29	373	4	table-lookup	11	0
30	377	1		3	9
31	378	2	no zeros, zerofill(1)	2	9
32	380	8	no zeros	2	9
33	388	4	always zero	2	9
34	392	4	table-lookup	12	0
35	396	4	no zeros, zerofill(1)	13	9
36	400	4	no zeros, zerofill(1)	2	9
37	404	1		2	9
38	405	4	no zeros	2	9
39	409	4	always zero	2	9
40	413	4	no zeros	2	9
41	417	4	always zero	2	9
42	421	4	no zeros	2	9
43	425	4	always zero	2	9
44	429	20	no empty	3	9
45	449	30	no empty	3	9
46	479	1		14	4
47	480	1		14	4
48	481	79	no endspace, no empty	15	9
49	560	79	no empty	2	9
50	639	79	no empty	2	9
51	718	79	no endspace	16	9
52	797	8	no empty	2	9
53	805	1		17	1
54	806	1		3	9
55	807	20	no empty	3	9
56	827	4	no zeros, zerofill(2)	2	9
57	831	4	no zeros, zerofill(1)	2	9

`myisampack` displays the following kinds of information:

- **normal**

The number of columns for which no extra packing is used.

- **empty-space**

The number of columns containing values that are only spaces. These occupy one bit.

- **empty-zero**

The number of columns containing values that are only binary zeros. These occupy one bit.

- **empty-fill**

The number of integer columns that do not occupy the full byte range of their type. These are changed to a smaller type. For example, a **BIGINT** column (eight bytes) can be stored as a **TINYINT** column (one byte) if all its values are in the range from **-128**

to 127.

- **pre-space**

The number of decimal columns that are stored with leading spaces. In this case, each value contains a count for the number of leading spaces.

- **end-space**

The number of columns that have a lot of trailing spaces. In this case, each value contains a count for the number of trailing spaces.

- **table-lookup**

The column had only a small number of different values, which were converted to an **ENUM** before Huffman compression.

- **zero**

The number of columns for which all values are zero.

- **Original trees**

The initial number of Huffman trees.

- **After join**

The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, **myisamchk -dvv** prints additional information about each column:

- **Type**

The data type. The value may contain any of the following descriptors:

- **constant**

All rows have the same value.

- **no endspace**

Do not store endspace.

- **no endspace, not_always**

Do not store endspace and do not do endspace compression for all values.

- **no endspace, no empty**

Do not store endspace. Do not store empty values.

- **table-lookup**

The column was converted to an **ENUM**.

- **zerofill(*N*)**

The most significant *N* bytes in the value are always 0 and are not stored.

- **no zeros**

Do not store zeros.

- **always zero**

Zero values are stored using one bit.

- **Huff tree**

The number of the Huffman tree associated with the column.

- **Bits**

The number of bits used in the Huffman tree.

After you run **myisampack**, you must run **myisamchk** to re-create any indexes. At this time, you can also sort the index blocks and create statistics needed for the MySQL optimizer to work more efficiently:

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

After you have installed the packed table into the MySQL database directory, you should execute **mysqladmin flush-tables** to force **mysqld** to start using the new table.

To unpack a packed table, use the **--unpack** option to **myisamchk**.

7.8. **mysql** — The MySQL Command-Line Tool

mysql is a simple SQL shell (with GNU **readline** capabilities). It supports interactive and non-interactive use. When used interactively, query results are presented in an ASCII-table format. When used non-interactively (for example, as a filter), the result is presented in tab-separated format. The output format can be changed using command options.

If you have problems due to insufficient memory for large result sets, use the **--quick** option. This forces **mysql** to retrieve results from the server a row at a time rather than retrieving the entire result set and buffering it in memory before displaying it. This is done by returning the result set using the **mysql_use_result()** C API function in the client/server library rather than **mysql_store_result()**.

Using **mysql** is very easy. Invoke it from the prompt of your command interpreter as follows:

```
shell> mysql db_name
```

Or:

```
shell> mysql --user=user_name --password=your_password db_name
```

Then type an SQL statement, end it with **;**, **\g**, or **\G** and press Enter.

As of MySQL 5.1.10, typing Control-C causes **mysql** to attempt to kill the current statement. If this cannot be done, or Control-C is typed again before the statement is killed, **mysql** exits. Previously, Control-C caused **mysql** to exit in all cases.

You can execute SQL statements in a script file (batch file) like this:

```
shell> mysql db_name < script.sql > output.tab
```

7.8.1. **mysql** Options

mysql supports the following options:

- **--help, -?**

Display a help message and exit.

- **--auto-rehash**

Enable automatic rehashing. This option is on by default, which enables table and column name completion. Use **-skip-auto-rehash** to disable rehashing. That causes **mysql** to start faster, but you must issue the **rehash** command if you want to use table and column name completion.

- **--batch, -B**

Print results using tab as the column separator, with each row on a new line. With this option, `mysql` does not use the history file.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--column-names`

Write column names in results.

- `--column-type-info, -m`

Display result set metadata. This option was added in MySQL 5.1.14. (Before that, use `--debug-info`.) The `-m` short option was added in MySQL 5.1.21.

- `--compress, -C`

Compress all information sent between the client and the server if both support compression.

- `--database=db_name, -D db_name`

The database to use. This is useful primarily in an option file.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The `debug_options` string often is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/mysql.trace'`.

- `--debug-check`

Print some debugging information when the program exits. This option was added in MySQL 5.1.21.

- `--debug-info, -T`

Before MySQL 5.1.14, this option prints debugging information and memory and CPU usage statistics when the program exits, and also causes display of result set metadata during execution. As of MySQL 5.1.14, use `--column-type-info` to display result set metadata.

- `--default-character-set=charset_name`

Use `charset_name` as the default character set. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--delimiter=str`

Set the statement delimiter. The default is the semicolon character (`;`).

- `--execute=statement, -e statement`

Execute the statement and quit. The default output format is like that produced with `--batch`. See [Section 4.3.1, “Using Options on the Command Line”](#), for some examples.

- `--force, -f`

Continue even if an SQL error occurs.

- `--host=host_name, -h host_name`

Connect to the MySQL server on the given host.

- `--html, -H`

Produce HTML output.

- `--ignore-spaces, -i`

Ignore spaces after function names. The effect of this is described in the discussion for the `IGNORE_SPACE` SQL mode (see [Sec-](#)

tion 5.2.6, “SQL Modes”).

- `--line-numbers`

Write line numbers for errors. Disable this with `--skip-line-numbers`.

- `--local-infile[={0|1}]`

Enable or disable **LOCAL** capability for **LOAD DATA INFILE**. With no value, the option enables **LOCAL**. The option may be given as `--local-infile=0` or `--local-infile=1` to explicitly disable or enable **LOCAL**. Enabling **LOCAL** has no effect if the server does not also support it.

MySQL Enterprise

For expert advice on the security implications of enabling **LOCAL**, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `--named-commands, -G`

Enable named **mysql** commands. Long-format commands are allowed, not just short-format commands. For example, **quit** and **\q** both are recognized. Use `--skip-named-commands` to disable named commands. See Section 7.8.2, “**mysql** Commands”.

- `--no-auto-rehash, -A`

Deprecated form of `--skip-auto-rehash`. See the description for `--auto-rehash`.

- `--no-beep, -b`

Do not beep when errors occur.

- `--no-named-commands, -g`

Disable named commands. Use the `\*` form only, or use named commands only at the beginning of a line ending with a semicolon (`;`). **mysql** starts with this option *enabled* by default. However, even with this option, long-format commands still work from the first line. See Section 7.8.2, “**mysql** Commands”.

- `--no-pager`

Deprecated form of `--skip-pager`. See the `--pager` option.

- `--no-tee`

Do not copy output to a file. Section 7.8.2, “**mysql** Commands”, discusses tee files further.

- `--one-database, -o`

Ignore statements except those for the default database named on the command line. This is useful for skipping updates to other databases in the binary log.

- `--pager[=command]`

Use the given command for paging query output. If the command is omitted, the default pager is the value of your **PAGER** environment variable. Valid pagers are **less**, **more**, **cat** [**>** *filename*], and so forth. This option works only on Unix. It does not work in batch mode. To disable paging, use `--skip-pager`. Section 7.8.2, “**mysql** Commands”, discusses output paging further.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See Section 5.8.6, “Keeping Your Password Secure”.

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--prompt=format_str`

Set the prompt to the specified format. The default is `mysql>`. The special sequences that the prompt can contain are described in [Section 7.8.2, “mysql Commands”](#).

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--quick, -q`

Do not cache each query result, print each row as it is received. This may slow down the server if the output is suspended. With this option, `mysql` does not use the history file.

- `--raw, -r`

Write column values without escape conversion. Often used with the `--batch` option.

- `--reconnect`

If the connection to the server is lost, automatically try to reconnect. A single reconnect attempt is made each time the connection is lost. To suppress reconnection behavior, use `--skip-reconnect`.

- `--safe-updates, --i-am-a-dummy, -U`

Allow only those `UPDATE` and `DELETE` statements that specify which rows to modify by using key values. If you have set this option in an option file, you can override it by using `--safe-updates` on the command line. See [Section 7.8.5, “mysql Tips”](#), for more information about this option.

- `--secure-auth`

Do not send passwords to the server in old (pre-4.1.1) format. This prevents connections except for servers that use the newer password format.

MySQL Enterprise

For expert advice on database security, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `--show-warnings`

Cause warnings to be shown after each statement if there are any. This option applies to interactive and batch mode.

- `--sigint-ignore`

Ignore `SIGINT` signals (typically the result of typing Control-C).

- `--silent, -s`

Silent mode. Produce less output. This option can be given multiple times to produce less and less output.

- `--skip-column-names, -N`

Do not write column names in results.

- `--skip-line-numbers, -L`

Do not write line numbers for errors. Useful when you want to compare result files that include error messages.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.8.7.3, “SSL Command Options”](#).

- `--table, -t`

Display output in table format. This is the default for interactive use, but can be used to produce table output in batch mode.

- `--tee=filename`

Append a copy of output to the given file. This option does not work in batch mode. in [Section 7.8.2, “mysql Commands”](#), discusses tee files further.

- `--unbuffered, -n`

Flush the buffer after each query.

- `--user=username, -u username`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Produce more output about what the program does. This option can be given multiple times to produce more and more output. (For example, `-v -v -v` produces table output format even in batch mode.)

- `--version, -V`

Display version information and exit.

- `--vertical, -E`

Print query output rows vertically (one line per column value). Without this option, you can specify vertical output for individual statements by terminating them with `\G`.

- `--wait, -w`

If the connection cannot be established, wait and retry instead of aborting.

- `--xml, -X`

Produce XML output.

Note

Prior to MySQL 5.1.12, there was no differentiation in the output when using this option between columns containing the `NULL` value and columns containing the string literal `'NULL'`; both were represented as

```
<field name="column_name">NULL</field>
```

Beginning with MySQL 5.1.12, the output when `--xml` is used with `mysql` matches that of `mysqldump --xml`. See [the section of the Manual which discusses the --xml option for mysqldump](#) for details.

Beginning with MySQL 5.1.18, the XML output also uses an XML namespace, as shown here:

```
shell> mysql --xml -uroot -e "SHOW VARIABLES LIKE 'version%'"
<?xml version="1.0"?>

<resultset statement="SHOW VARIABLES LIKE 'version%'" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <field name="Variable_name">version</field>
    <field name="Value">5.0.40-debug</field>
  </row>

  <row>
    <field name="Variable_name">version_comment</field>
    <field name="Value">Source distribution</field>
  </row>

  <row>
    <field name="Variable_name">version_compile_machine</field>
    <field name="Value">i686</field>
  </row>

  <row>
    <field name="Variable_name">version_compile_os</field>
```

```
<field name="Value">suse-linux-gnu</field>
</row>
</resultset>
```

(See [Bug#25946](#).)

You can also set the following variables by using `--var_name=value` syntax:

- `connect_timeout`

The number of seconds before connection timeout. (Default value is 0.)

- `max_allowed_packet`

The maximum packet length to send to or receive from the server. (Default value is 16MB.)

- `max_join_size`

The automatic limit for rows in a join when using `--safe-updates`. (Default value is 1,000,000.)

- `net_buffer_length`

The buffer size for TCP/IP and socket communication. (Default value is 16KB.)

- `select_limit`

The automatic limit for `SELECT` statements when using `--safe-updates`. (Default value is 1,000.)

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated.*

On Unix, the `mysql` client writes a record of executed statements to a history file. By default, the history file is named `.mysql_history` and is created in your home directory. To specify a different file, set the value of the `MYSQL_HISTFILE` environment variable.

If you do not want to maintain a history file, first remove `.mysql_history` if it exists, and then use either of the following techniques:

- Set the `MYSQL_HISTFILE` variable to `/dev/null`. To cause this setting to take effect each time you log in, put the setting in one of your shell's startup files.
- Create `.mysql_history` as a symbolic link to `/dev/null`:

```
shell> ln -s /dev/null $HOME/.mysql_history
```

You need do this only once.

7.8.2. `mysql` Commands

`mysql` sends each SQL statement that you issue to the server to be executed. There is also a set of commands that `mysql` itself interprets. For a list of these commands, type `help` or `\h` at the `mysql>` prompt:

```
mysql> help
List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?          (\?) Synonym for 'help'.
charset    (\C) Switch to another charset. Might be needed for processing
            binlog with multi-byte charsets.
clear      (\c) Clear command.
connect    (\r) Reconnect to the server. Optional arguments are db and host.
delimiter  (\d) Set statement delimiter. NOTE: Takes the rest of the line as
            new delimiter.
edit       (\e) Edit command with $EDITOR.
ego        (\G) Send command to mysql server, display result vertically.
```

```

exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source     (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
system    (\!) Execute a system shell command.
tee        (\T) Set outfile [to_outfile]. Append everything into given
           outfile.
use        (\u) Use another database. Takes database name as argument.
warnings  (\W) Show warnings after every statement.
nowarning  (\w) Don't show warnings after every statement.

For server side help, type 'help contents'

```

Each command has both a long and short form. The long form is not case sensitive; the short form is. The long form can be followed by an optional semicolon terminator, but the short form should not.

If you provide an argument to the `help` command, `mysql` uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. For more information, see [Section 7.8.3, “mysql Server-Side Help”](#).

The `charset` command changes the default character set and issues a `SET NAMES` statement. This enables the character set to remain synchronized on the client and server if `mysql` is run with auto-reconnect enabled (which is not recommended), because the changed character set is used for reconnects. This command was added in MySQL 5.1.12.

In the `delimiter` command, you should avoid the use of the backslash (`\`) character because that is the escape character for MySQL.

The `edit`, `nopager`, `pager`, and `system` commands work only in Unix.

The `status` command provides some information about the connection and the server you are using. If you are running in `-safe-updates` mode, `status` also prints the values for the `mysql` variables that affect your queries.

To log queries and their output, use the `tee` command. All the data displayed on the screen is appended into a given file. This can be very useful for debugging purposes also. You can enable this feature on the command line with the `--tee` option, or interactively with the `tee` command. The `tee` file can be disabled interactively with the `notee` command. Executing `tee` again re-enables logging. Without a parameter, the previous file is used. Note that `tee` flushes query results to the file after each statement, just before `mysql` prints its next prompt.

By using the `--pager` option, it is possible to browse or search query results in interactive mode with Unix programs such as `less`, `more`, or any other similar program. If you specify no value for the option, `mysql` checks the value of the `PAGER` environment variable and sets the pager to that. Output paging can be enabled interactively with the `pager` command and disabled with `nopager`. The command takes an optional argument; if given, the paging program is set to that. With no argument, the pager is set to the pager that was set on the command line, or `stdout` if no pager was specified.

Output paging works only in Unix because it uses the `popen()` function, which does not exist on Windows. For Windows, the `tee` option can be used instead to save query output, although this is not as convenient as `pager` for browsing output in some situations.

Here are a few tips about the `pager` command:

- You can use it to write to a file and the results go only to the file:

```
mysql> pager cat > /tmp/log.txt
```

You can also pass any options for the program that you want to use as your pager:

```
mysql> pager less -n -i -S
```

- In the preceding example, note the `-S` option. You may find it very useful for browsing wide query results. Sometimes a very wide result set is difficult to read on the screen. The `-S` option to `less` can make the result set much more readable because you can scroll it horizontally using the left-arrow and right-arrow keys. You can also use `-S` interactively within `less` to switch the horizontal-browse mode on and off. For more information, read the `less` manual page:

```
shell> man less
```

- You can specify very complex pager commands for handling query output:

```
mysql> pager cat | tee /dr1/tmp/res.txt \
      | tee /dr2/tmp/res2.txt | less -n -i -S
```

In this example, the command would send query results to two files in two different directories on two different filesystems mounted on `/dr1` and `/dr2`, yet still display the results onscreen via `less`.

You can also combine the `tee` and `pager` functions. Have a `tee` file enabled and `pager` set to `less`, and you are able to browse the results using the `less` program and still have everything appended into a file the same time. The difference between the Unix `tee` used with the `pager` command and the `mysql` built-in `tee` command is that the built-in `tee` works even if you do not have the Unix `tee` available. The built-in `tee` also logs everything that is printed on the screen, whereas the Unix `tee` used with `pager` does not log quite that much. Additionally, `tee` file logging can be turned on and off interactively from within `mysql`. This is useful when you want to log some queries to a file, but not others.

The default `mysql>` prompt can be reconfigured. The string for defining the prompt can contain the following special sequences:

Option	Description
<code>\v</code>	The server version
<code>\d</code>	The default database
<code>\h</code>	The server host
<code>\p</code>	The current TCP/IP port or socket file
<code>\u</code>	Your username
<code>\U</code>	Your full <code>user_name@host_name</code> account name
<code>\\</code>	A literal ‘\’ backslash character
<code>\n</code>	A newline character
<code>\t</code>	A tab character
<code>\ </code>	A space (a space follows the backslash)
<code>_</code>	A space
<code>\R</code>	The current time, in 24-hour military time (0-23)
<code>\r</code>	The current time, standard 12-hour time (1-12)
<code>\m</code>	Minutes of the current time
<code>\y</code>	The current year, two digits
<code>\Y</code>	The current year, four digits
<code>\D</code>	The full current date
<code>\s</code>	Seconds of the current time
<code>\w</code>	The current day of the week in three-letter format (Mon, Tue, ...)
<code>\P</code>	am/pm
<code>\o</code>	The current month in numeric format
<code>\O</code>	The current month in three-letter format (Jan, Feb, ...)
<code>\c</code>	A counter that increments for each statement you issue
<code>\l</code>	The current delimiter. (New in 5.1.12)
<code>\S</code>	Semicolon
<code>\'</code>	Single quote
<code>\"</code>	Double quote

‘\’ followed by any other letter just becomes that letter.

If you specify the `prompt` command with no argument, `mysql` resets the prompt to the default of `mysql>`.

You can set the prompt in several ways:

- *Use an environment variable.* You can set the `MYSQL_PS1` environment variable to a prompt string. For example:

```
shell> export MYSQL_PS1="(\\u@\\h) [\\d]> "
```

- *Use a command-line option.* You can set the `--prompt` option on the command line to `mysql`. For example:

```
shell> mysql --prompt="(\\u@\\h) [\\d]> "
(user@host) [database]>
```

- *Use an option file.* You can set the `prompt` option in the `[mysql]` group of any MySQL option file, such as `/etc/my.cnf` or the `.my.cnf` file in your home directory. For example:

```
[mysql]
prompt=(\\u@\\h) [\\d]>\\_
```

In this example, note that the backslashes are doubled. If you set the prompt using the `prompt` option in an option file, it is advisable to double the backslashes when using the special prompt options. There is some overlap in the set of allowable prompt options and the set of special escape sequences that are recognized in option files. (These sequences are listed in [Section 4.3.2, “Using Option Files”](#).) The overlap may cause you problems if you use single backslashes. For example, `\\s` is interpreted as a space rather than as the current seconds value. The following example shows how to define a prompt within an option file to include the current time in `HH:MM:SS>` format:

```
[mysql]
prompt="\\r:\\m:\\s> "
```

- *Set the prompt interactively.* You can change your prompt interactively by using the `prompt` (or `\\R`) command. For example:

```
mysql> prompt (\\u@\\h) [\\d]>\\_
PROMPT set to '(\\u@\\h) [\\d]>\\_'
(user@host) [database]>
(user@host) [database]> prompt
Returning to default PROMPT of mysql>
mysql>
```

7.8.3. `mysql` Server-Side Help

```
mysql> help search_string
```

If you provide an argument to the `help` command, `mysql` uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. The proper operation of this command requires that the help tables in the `mysql` database be initialized with help topic information (see [Section 5.2.8, “Server-Side Help”](#)).

If there is no match for the search string, the search fails:

```
mysql> help me
Nothing found
Please try to run 'help contents' for a list of all accessible topics
```

Use `help contents` to see a list of the help categories:

```
mysql> help contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the
following categories:
Account Management
Administration
Data Definition
Data Manipulation
Data Types
Functions
Functions and Modifiers for Use with GROUP BY
Geographic Features
Language Structure
```



```

Plugins
Storage Engines
Stored Routines
Table Maintenance
Transactions
Triggers

```

If the search string matches multiple items, `mysql` shows a list of matching topics:

```

mysql> help logs
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following topics:
  SHOW
  SHOW BINARY LOGS
  SHOW ENGINE
  SHOW LOGS

```

Use a topic as the search string to see the help entry for that topic:

```

mysql> help show binary logs
Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS

Lists the binary log files on the server. This statement is used as
part of the procedure described in [purge-master-logs], that shows how
to determine which logs can be purged.

mysql> SHOW BINARY LOGS;
+-----+-----+
| Log_name          | File_size |
+-----+-----+
| binlog.000015     | 724935   |
| binlog.000016     | 733481   |
+-----+-----+

```

7.8.4. Executing SQL Statements from a Text File

The `mysql` client typically is used interactively, like this:

```
shell> mysql db_name
```

However, it is also possible to put your SQL statements in a file and then tell `mysql` to read its input from that file. To do so, create a text file `text_file` that contains the statements you wish to execute. Then invoke `mysql` as shown here:

```
shell> mysql db_name < text_file
```

If you place a `USE db_name` statement as the first statement in the file, it is unnecessary to specify the database name on the command line:

```
shell> mysql < text_file
```

If you are already running `mysql`, you can execute an SQL script file using the `source` command or `\.` command:

```

mysql> source file_name
mysql> \. file_name

```

Sometimes you may want your script to display progress information to the user. For this you can insert statements like this:

```
SELECT '<info_to_display>' AS ' ';
```

The statement shown outputs `<info_to_display>`.

For more information about batch mode, see [Section 3.5, “Using mysql in Batch Mode”](#).

7.8.5. mysql Tips

This section describes some techniques that can help you use `mysql` more effectively.

7.8.5.1. Displaying Query Results Vertically

Some query results are much more readable when displayed vertically, instead of in the usual horizontal table format. Queries can be displayed vertically by terminating the query with `\G` instead of a semicolon. For example, longer text values that include newlines often are much easier to read with vertical output:

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,\G
***** 1. row *****
  msg_nro: 3068
    date: 2000-03-01 23:29:50
  time_zone: +0200
  mail_from: Monty
    reply: monty@no.spam.com
  mail_to: "Thimble Smith" <tim@no.spam.com>
    sbj: UTF-8
    txt: >>>> "Thimble" == Thimble Smith writes:

Thimble> Hi. I think this is a good idea. Is anyone familiar
Thimble> with UTF-8 or Unicode? Otherwise, I'll put this on my
Thimble> TODO list and see what happens.

Yes, please do that.

Regards,
Monty
      file: inbox-jani-1
      hash: 190402944
1 row in set (0.09 sec)
```

7.8.5.2. Using the `--safe-updates` Option

For beginners, a useful startup option is `--safe-updates` (or `--i-am-a-dummy`, which has the same effect). It is helpful for cases when you might have issued a `DELETE FROM tbl_name` statement but forgotten the `WHERE` clause. Normally, such a statement deletes all rows from the table. With `--safe-updates`, you can delete rows only by specifying the key values that identify them. This helps prevent accidents.

When you use the `--safe-updates` option, `mysql` issues the following statement when it connects to the MySQL server:

```
SET SQL_SAFE_UPDATES=1,SQL_SELECT_LIMIT=1000, SQL_MAX_JOIN_SIZE=1000000;
```

See [Section 12.5.3, “SET Syntax”](#).

The `SET` statement has the following effects:

- You are not allowed to execute an `UPDATE` or `DELETE` statement unless you specify a key constraint in the `WHERE` clause or provide a `LIMIT` clause (or both). For example:

```
UPDATE tbl_name SET not_key_column=val WHERE key_column=val;
UPDATE tbl_name SET not_key_column=val LIMIT 1;
```

- The server limits all large `SELECT` results to 1,000 rows unless the statement includes a `LIMIT` clause.
- The server aborts multiple-table `SELECT` statements that probably need to examine more than 1,000,000 row combinations.

To specify limits different from 1,000 and 1,000,000, you can override the defaults by using the `--select_limit` and `--max_join_size` options:

```
shell> mysql --safe-updates --select_limit=500 --max_join_size=10000
```

7.8.5.3. Disabling `mysql` Auto-Reconnect

If the `mysql` client loses its connection to the server while sending a statement, it immediately and automatically tries to reconnect once to the server and send the statement again. However, even if `mysql` succeeds in reconnecting, your first connection has ended and all your previous session objects and settings are lost: temporary tables, the autocommit mode, and user-defined and session variables.

Also, any current transaction rolls back. This behavior may be dangerous for you, as in the following example where the server was shut down and restarted between the first and second statements without you knowing it:

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id: 1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
+-----+
| a     |
+-----+
| NULL  |
+-----+
1 row in set (0.05 sec)
```

The `@a` user variable has been lost with the connection, and after the reconnection it is undefined. If it is important to have `mysql` terminate with an error if the connection has been lost, you can start the `mysql` client with the `--skip-reconnect` option.

For more information about auto-reconnect and its effect on state information when a reconnection occurs, see [Section 25.2.13](#), “Controlling Automatic Reconnect Behavior”.

7.9. `mysqlaccess` — Client for Checking Access Privileges

`mysqlaccess` is a diagnostic tool that Yves Carlier has provided for the MySQL distribution. It checks the access privileges for a hostname, username, and database combination. Note that `mysqlaccess` checks access using only the `user`, `db`, and `host` tables. It does not check table, column, or routine privileges specified in the `tables_priv`, `columns_priv`, or `procs_priv` tables.

Invoke `mysqlaccess` like this:

```
shell> mysqlaccess [host_name [user_name [db_name]]] [options]
```

`mysqlaccess` understands the following options:

- `--help, -?`
Display a help message and exit.
- `--brief, -b`
Generate reports in single-line tabular format.
- `--commit`
Copy the new access privileges from the temporary tables to the original grant tables. The grant tables must be flushed for the new privileges to take effect. (For example, execute a `mysqladmin reload` command.)
- `--copy`
Reload the temporary grant tables from original ones.
- `--db=db_name, -d db_name`
Specify the database name.
- `--debug=N`
Specify the debug level. *N* can be an integer from 0 to 3.
- `--host=host_name, -h host_name`
The hostname to use in the access privileges.

- `--howto`

Display some examples that show how to use `mysqlaccess`.

- `--old_server`

Assume that the server is an old MySQL server (before MySQL 3.21) that does not yet know how to handle full `WHERE` clauses.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--plan`

Display suggestions and ideas for future releases.

- `--preview`

Show the privilege differences after making changes to the temporary grant tables.

- `--relnotes`

Display the release notes.

- `--rhost=host_name, -H host_name`

Connect to the MySQL server on the given host.

- `--rollback`

Undo the most recent changes to the temporary grant tables.

- `--spassword[=password], -P[password]`

The password to use when connecting to the server as the superuser. If you omit the *password* value following the `--password` or `-P` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--superuser=user_name, -U user_name`

Specify the username for connecting as the superuser.

- `--table, -t`

Generate reports in table format.

- `--user=user_name, -u user_name`

The username to use in the access privileges.

- `--version, -v`

Display version information and exit.

If your MySQL distribution is installed in some non-standard location, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the `mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL = '/usr/local/bin/mysql'; # path to mysql executable
```

Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, a `Broken pipe` error will occur when you run `mysqlaccess`.

7.10. `mysqladmin` — Client for Administering a MySQL Server

`mysqladmin` is a client for performing administrative operations. You can use it to check the server's configuration and current status, to create and drop databases, and more.

Invoke `mysqladmin` like this:

```
shell> mysqladmin [options] command [command-arg] [command [command-arg]] ...
```

`mysqladmin` supports the commands described in the following list. Some of the commands take an argument following the command name.

- `create db_name`

Create a new database named `db_name`.

- `debug`

Tell the server to write debug information to the error log.

Beginning with MySQL 5.1.12, this includes information about the Event Scheduler. See [Section 21.4, “Event Scheduler Status”](#).

- `--debug-check`

Print some debugging information when the program exits. This option was added in MySQL 5.1.21.

- `--debug-info`

Print debugging information and memory and CPU usage statistics when the program exits. This option was added in MySQL 5.1.21.

- `drop db_name`

Delete the database named `db_name` and all its tables.

- `extended-status`

Display the server status variables and their values.

MySQL Enterprise

For expert advice on using server status variables, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `flush-hosts`

Flush all information in the host cache.

- `flush-logs`

Flush all logs.

- `flush-privileges`

Reload the grant tables (same as `reload`).

- `flush-status`

Clear status variables.

- `flush-tables`

Flush all tables.

- `flush-threads`

Flush the thread cache.

- `kill id,id,...`

Kill server threads. If multiple thread ID values are given, there must be no spaces in the list.

- `old-password new-password`

This is like the `password` command but stores the password using the old (pre-4.1) password-hashing format. (See [Section 5.7.9](#), “Password Hashing as of MySQL 4.1”.)

MySQL Enterprise

For expert advice on the security implications of using the `old-password` command, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `password new-password`

Set a new password. This changes the password to `new-password` for the account that you use with `mysqladmin` for connecting to the server. Thus, the next time you invoke `mysqladmin` (or any other client program) using the same account, you will need to specify the new password.

If the `new-password` value contains spaces or other characters that are special to your command interpreter, you need to enclose it within quotes. On Windows, be sure to use double quotes rather than single quotes; single quotes are not stripped from the password, but rather are interpreted as part of the password. For example:

```
shell> mysqladmin password "my new password"
```

- `ping`

Check whether the server is alive. The return status from `mysqladmin` is 0 if the server is running, 1 if it is not. This is 0 even in case of an error such as `Access denied`, because this means that the server is running but refused the connection, which is different from the server not running.

- `processlist`

Show a list of active server threads. This is like the output of the `SHOW PROCESSLIST` statement. If the `--verbose` option is given, the output is like that of `SHOW FULL PROCESSLIST`. (See [Section 12.5.4.25](#), “`SHOW PROCESSLIST` Syntax”.)

- `reload`

Reload the grant tables.

- `refresh`

Flush all tables and close and open log files.

- `shutdown`

Stop the server.

- `start-slave`

Start replication on a slave server.

- `status`

Display a short server status message.

- `stop-slave`

Stop replication on a slave server.

- `variables`

Display the server system variables and their values.

MySQL Enterprise

For expert advice on using server system variables, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- **version**

Display version information from the server.

All commands can be shortened to any unique prefix. For example:

```
shell> mysqladmin proc stat
+-----+-----+-----+-----+-----+-----+-----+-----+
| Id | User | Host | db | Command | Time | State | Info |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 51 | monty | localhost | | Query | 0 | | show processlist |
+-----+-----+-----+-----+-----+-----+-----+-----+
Uptime: 1473624 Threads: 1 Questions: 39487
Slow queries: 0 Opens: 541 Flush tables: 1
Open tables: 19 Queries per second avg: 0.0268
```

The `mysqladmin status` command result displays the following values:

- **Uptime**

The number of seconds the MySQL server has been running.

- **Threads**

The number of active threads (clients).

- **Questions**

The number of questions (queries) from clients since the server was started.

- **Slow queries**

The number of queries that have taken more than `long_query_time` seconds. See [Section 5.11.5, “The Slow Query Log”](#).

- **Opens**

The number of tables the server has opened.

- **Flush tables**

The number of `flush-*`, `refresh`, and `reload` commands the server has executed.

- **Open tables**

The number of tables that currently are open.

- **Memory in use**

The amount of memory allocated directly by `mysqld`. This value is displayed only when MySQL has been compiled with `--with-debug=full`.

- **Maximum memory used**

The maximum amount of memory allocated directly by `mysqld`. This value is displayed only when MySQL has been compiled with `--with-debug=full`.

If you execute `mysqladmin shutdown` when connecting to a local server using a Unix socket file, `mysqladmin` waits until the server's process ID file has been removed, to ensure that the server has stopped properly.

`mysqladmin` supports the following options:

- `--help, -?`
Display a help message and exit.
- `--character-sets-dir=path`
The directory where character sets are installed. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).
- `--compress, -C`
Compress all information sent between the client and the server if both support compression.
- `--count=N, -c N`
The number of iterations to make for repeated command execution. This works only with the `--sleep` option.
- `--debug[=debug_options], -# [debug_options]`
Write a debugging log. The *debug_options* string often is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/mysqladmin.trace'`.
- `--default-character-set=charset_name`
Use *charset_name* as the default character set. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).
- `--force, -f`
Do not ask for confirmation for the `drop db_name` command. With multiple commands, continue even if an error occurs.
- `--host=host_name, -h host_name`
Connect to the MySQL server on the given host.
- `--no-beep, -b`
Suppress the warning beep that is emitted by default for errors such as a failure to connect to the server. This option was added in MySQL 5.1.17.
- `--password[=password], -p[password]`
The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).
- `--port=port_num, -P port_num`
The TCP/IP port number to use for the connection.
- `--protocol={TCP|SOCKET|PIPE|MEMORY}`
The connection protocol to use.
- `--relative, -r`
Show the difference between the current and previous values when used with the `--sleep` option. Currently, this option works only with the `extended-status` command.
- `--silent, -s`
Exit silently if a connection to the server cannot be established.
- `--sleep=delay, -i delay`
Execute commands repeatedly, sleeping for *delay* seconds in between. The `--count` option determines the number of iterations.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.8.7.3, “SSL Command Options”](#).

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print more information about what the program does.

- `--version, -V`

Display version information and exit.

- `--vertical, -E`

Print output vertically. This is similar to `--relative`, but prints output vertically.

- `--wait[=count], -w[count]`

If the connection cannot be established, wait and retry instead of aborting. If a *count* value is given, it indicates the number of times to retry. The default is one time.

You can also set the following variables by using `--var_name=value` syntax:

- `connect_timeout`

The maximum number of seconds before connection timeout. The default value is 43200 (12 hours).

- `shutdown_timeout`

The maximum number of seconds to wait for server shutdown. The default value is 3600 (1 hour).

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated.*

7.11. `mysqlbinlog` — Utility for Processing Binary Log Files

The binary log files that the server generates are written in binary format. To examine these files in text format, use the `mysqlbinlog` utility. You can also use `mysqlbinlog` to read relay log files written by a slave server in a replication setup. Relay logs have the same format as binary log files.

Invoke `mysqlbinlog` like this:

```
shell> mysqlbinlog [options] log_file ...
```

For example, to display the contents of the binary log file named `binlog.000003`, use this command:

```
shell> mysqlbinlog binlog.000003
```

The output includes all events contained in `binlog.000003`. Event information includes the statement executed, the time the statement took, the thread ID of the client that issued it, the timestamp when it was executed, and so forth.

The output from `mysqlbinlog` can be re-executed (for example, by using it as input to `mysql`) to reapply the statements in the log. This is useful for recovery operations after a server crash. For other usage examples, see the discussion later in this section.

Normally, you use `mysqlbinlog` to read binary log files directly and apply them to the local MySQL server. It is also possible to read binary logs from a remote server by using the `--read-from-remote-server` option. When you read remote binary logs, the connection parameter options can be given to indicate how to connect to the server. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`; they are ignored except when you also use the `--read-from-remote-server` option.

Binary logs and relay logs are discussed further in [Section 5.11.4, “The Binary Log”](#), and [Section 15.4.2, “Replication Relay and Status Files”](#).

`mysqlbinlog` supports the following options:

- `--help, -?`

Display a help message and exit.

- `--base64-output`

Print all binary log entries using base64 encoding. This is for debugging only. Logs produced using this option should not be applied on production systems. This option was added in MySQL 5.1.5.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--database=db_name, -d db_name`

List entries for just this database (local log only). You can only specify one database with this option - if you specify multiple `--database` options, only the last one is used. This option forces `mysqlbinlog` to output entries from the binary log where the default database (that is, the one selected by `USE`) is `db_name`. Note that this does not replicate cross-database statements such as `UPDATE some_db.some_table SET foo='bar'` while having selected a different database or no database.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. A typical `debug_options` string is often `'d:t:o,file_name'`.

- `--debug-check`

Print some debugging information when the program exits. This option was added in MySQL 5.1.21.

- `--debug-info`

Print debugging information and memory and CPU usage statistics when the program exits. This option was added in MySQL 5.1.21.

- `--disable-log-bin, -D`

Disable binary logging. This is useful for avoiding an endless loop if you use the `--to-last-log` option and are sending the output to the same MySQL server. This option also is useful when restoring after a crash to avoid duplication of the statements you have logged.

This option requires that you have the `SUPER` privilege. It causes `mysqlbinlog` to include a `SET SQL_LOG_BIN=0` statement in its output to disable binary logging of the remaining output. The `SET` statement is ineffective unless you have the `SUPER` privilege.

- `--force-read, -f`

With this option, if `mysqlbinlog` reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this option, `mysqlbinlog` stops if it reads such an event.

- `--hexdump, -H`

Display a hex dump of the log in comments. This output can be helpful for replication debugging. Hex dump format is discussed later in this section. This option was added in MySQL 5.1.2.

- `--host=host_name, -h host_name`

Get the binary log from the MySQL server on the given host.

- `--local-load=path, -l path`

Prepare local temporary files for `LOAD DATA INFILE` in the specified directory.

- `--offset=N, -o N`

Skip the first *N* entries in the log.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for connecting to a remote server.

- `--position=N, -j N`

Deprecated. Use `--start-position` instead.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--read-from-remote-server, -R`

Read the binary log from a MySQL server rather than reading a local log file. Any connection parameter options are ignored unless this option is given as well. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`.

- `--result-file=name, -r name`

Direct output to the given file.

- `--server-id=id`

Extract only those events created by the server having the given server ID. This option is available as of MySQL 5.1.4.

- `--set-charset=charset_name`

Add a `SET NAMES charset_name` statement to the output to specify the character set to be used for processing log files. This option was added in MySQL 5.1.12.

- `--short-form, -s`

Display only the statements contained in the log, without any extra information.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--start-datetime=datetime`

Start reading the binary log at the first event having a timestamp equal to or later than the *datetime* argument. The *datetime* value is relative to the local time zone on the machine where you run `mysqlbinlog`. The value should be in a format accepted for the `DATETIME` or `TIMESTAMP` data types. For example:

```
shell> mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003
```

This option is useful for point-in-time recovery. See [Section 5.9.2, “Example Backup and Recovery Strategy”](#).

- `--stop-datetime=datetime`

Stop reading the binary log at the first event having a timestamp equal or posterior to the `datetime` argument. This option is useful for point-in-time recovery. See the description of the `--start-datetime` option for information about the `datetime` value.

- `--start-position=N`

Start reading the binary log at the first event having a position equal to the `N` argument. This option applies to the first log file named on the command line.

- `--stop-position=N`

Stop reading the binary log at the first event having a position equal or greater than the `N` argument. This option applies to the last log file named on the command line.

- `--to-last-log, -t`

Do not stop at the end of the requested binary log from a MySQL server, but rather continue printing until the end of the last binary log. If you send the output to the same MySQL server, this may lead to an endless loop. This option requires `--read-from-remote-server`.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to a remote server.

- `--version, -V`

Display version information and exit.

- `--write-binlog`

This option is enabled by default, so that `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements generated by `mysqlcheck` are written to the binary log. Use `--skip-write-binlog` to cause `NO_WRITE_TO_BINLOG` to be added to the statements so that they are not logged. Use the `--skip-write-binlog` when these statements should not be sent to replication slaves or run when using the binary logs for recovery from backup. This option was added in MySQL 5.1.18.

You can also set the following variable by using `--var_name=value` syntax:

- `open_files_limit`

Specify the number of open file descriptors to reserve.

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated.*

You can pipe the output of `mysqlbinlog` into the `mysql` client to execute the statements contained in the binary log. This is used to recover from a crash when you have an old backup (see [Section 5.9.1, “Database Backups”](#)). For example:

```
shell> mysqlbinlog binlog.000001 | mysql
```

Or:

```
shell> mysqlbinlog binlog.[0-9]* | mysql
```

You can also redirect the output of `mysqlbinlog` to a text file instead, if you need to modify the statement log first (for example, to remove statements that you do not want to execute for some reason). After editing the file, execute the statements that it contains by using it as input to the `mysql` program.

`mysqlbinlog` has the `--start-position` option, which prints only those statements with an offset in the binary log greater than or equal to a given position (the given position must match the start of one event). It also has options to stop and start when it sees an event with a given date and time. This enables you to perform point-in-time recovery using the `--stop-datetime` option (to be able

to say, for example, “roll forward my databases to how they were today at 10:30 a.m.”).

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql # DANGER!!
```

Processing binary logs this way using different connections to the server causes problems if the first log file contains a `CREATE TEMPORARY TABLE` statement and the second log contains a statement that uses the temporary table. When the first `mysql` process terminates, the server drops the temporary table. When the second `mysql` process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* connection to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 > /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -e "source /tmp/statements.sql"
```

`mysqlbinlog` can produce output that reproduces a `LOAD DATA INFILE` operation without the original data file. `mysqlbinlog` copies the data to a temporary file and writes a `LOAD DATA LOCAL INFILE` statement that refers to the file. The default location of the directory where these files are written is system-specific. To specify a directory explicitly, use the `--local-load` option.

Because `mysqlbinlog` converts `LOAD DATA INFILE` statements to `LOAD DATA LOCAL INFILE` statements (that is, it adds `LOCAL`), both the client and the server that you use to process the statements must be configured to allow `LOCAL` capability. See [Section 5.6.4, “Security Issues with LOAD DATA LOCAL”](#).

MySQL Enterprise

For expert advice on the security implications of enabling `LOCAL`, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

Warning

The temporary files created for `LOAD DATA LOCAL` statements are *not* automatically deleted because they are needed until you actually execute those statements. You should delete the temporary files yourself after you no longer need the statement log. The files can be found in the temporary file directory and have names like `original_file_name-#-#`.

The `--hexdump` option produces a hex dump of the log contents in comments:

```
shell> mysqlbinlog --hexdump master-bin.000001
```

With the preceding command, the output might look like this:

```
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @@OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
# at 4
#051024 17:24:13 server id 1 end_log_pos 98
# Position Timestamp Type Master ID Size Master Pos Flags
# 00000004 9d fc 5c 43 0f 01 00 00 00 5e 00 00 00 62 00 00 00 00 00
# 00000017 04 00 35 2e 30 2e 31 35 2d 64 65 62 75 67 2d 6c |..5.0.15.debug.l|
# 00000027 6f 67 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |og.....|
# 00000037 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
# 00000047 00 00 00 00 00 9d fc 5c 43 13 38 0d 00 08 00 12 00 |.....C.8.....|
# 00000057 04 04 04 04 12 00 00 4b 00 04 1a |.....K...|
# Start: binlog v 4, server v 5.0.15-debug-log created 051024 17:24:13
# at startup
ROLLBACK;
```

Hex dump output currently contains the following elements. This format might change in the future.

- **Position:** The byte position within the log file.
- **Timestamp:** The event timestamp. In the example shown, '9d fc 5c 43' is the representation of '051024 17:24:13' in

hexadecimal.

- **Type:** The type of the log event. In the example shown, '0f' means that the example event is a `FORMAT_DESCRIPTION_EVENT`. The following table lists the possible types.

Type	Name	Meaning
00	UNKNOWN_EVENT	This event should never be present in the log.
01	START_EVENT_V3	This indicates the start of a log file written by MySQL 4 or earlier.
02	QUERY_EVENT	The most common type of events. These contain statements executed on the master.
03	STOP_EVENT	Indicates that master has stopped.
04	ROTATE_EVENT	Written when the master switches to a new log file.
05	INTVAR_EVENT	Used mainly for <code>AUTO_INCREMENT</code> values and when the <code>LAST_INSERT_ID()</code> function is used in the statement.
06	LOAD_EVENT	Used for <code>LOAD DATA INFILE</code> in MySQL 3.23.
07	SLAVE_EVENT	Reserved for future use.
08	CREATE_FILE_EVENT	Used for <code>LOAD DATA INFILE</code> statements. This indicates the start of execution of such a statement. A temporary file is created on the slave. Used in MySQL 4 only.
09	AP-PEND_BLOCK_EVENT	Contains data for use in a <code>LOAD DATA INFILE</code> statement. The data is stored in the temporary file on the slave.
0a	EXEC_LOAD_EVENT	Used for <code>LOAD DATA INFILE</code> statements. The contents of the temporary file is stored in the table on the slave. Used in MySQL 4 only.
0b	DELETE_FILE_EVENT	Rollback of a <code>LOAD DATA INFILE</code> statement. The temporary file should be deleted on slave.
0c	NEW_LOAD_EVENT	Used for <code>LOAD DATA INFILE</code> in MySQL 4 and earlier.
0d	RAND_EVENT	Used to send information about random values if the <code>RAND()</code> function is used in the statement.
0e	USER_VAR_EVENT	Used to replicate user variables.
0f	FORMAT_DESCRIPTION_EVENT	This indicates the start of a log file written by MySQL 5 or later.
10	XID_EVENT	Event indicating commit of an XA transaction.
11	BE-GIN_LOAD_QUERY_EVENT	Used for <code>LOAD DATA INFILE</code> statements in MySQL 5 and later.
12	EX-ECUTE_LOAD_QUERY_EVENT	Used for <code>LOAD DATA INFILE</code> statements in MySQL 5 and later.
13	TABLE_MAP_EVENT	Information about a table definition. Used in MySQL 5.1 and later.
14	WRITE_ROWS_EVENT	Row data for a single table that should be created. Used in MySQL 5.1 and later.
15	UPDATE_ROWS_EVENT	Row data for a single table that needs to be updated. Used in MySQL 5.1 and later.
16	DELETE_ROWS_EVENT	Row data for a single table that should be deleted. Used in MySQL 5.1 and later.

- **Master ID:** The server id of the master that created the event.
- **Size:** The size in bytes of the event.
- **Master Pos:** The position of the event in the original master log file.
- **Flags:** 16 flags. Currently, the following flags are used. The others are reserved for the future.

Flag	Name	Meaning
01	LOG_EVENT_BINLOG_IN_USE_F	Log file correctly closed. (Used only in <code>FORMAT_DESCRIPTION_EVENT</code> .) If this flag is set (if the flags are, for example, '01 00') in a <code>FORMAT_DESCRIPTION_EVENT</code> , the log file has not been properly closed. Most probably this is because of a master crash (for example, due to power failure).
02		Reserved for future use.

04	LOG_EVENT_THREAD_SPECIFIC_F	Set if the event is dependent on the connection it was executed in (for example, '04 00'), for example, if the event uses temporary tables.
08	LOG_EVENT_SUPPRESS_USE_F	Set in some circumstances when the event is not dependent on the default database.

The other flags are reserved for future use.

7.12. `mysqlcheck` — A Table Maintenance and Repair Program

The `mysqlcheck` client checks, repairs, optimizes, and analyzes tables.

`mysqlcheck` is similar in function to `myisamchk`, but works differently. The main operational difference is that `mysqlcheck` must be used when the `mysqld` server is running, whereas `myisamchk` should be used when it is not. The benefit of using `mysqlcheck` is that you do not have to stop the server to check or repair your tables.

`mysqlcheck` uses the SQL statements `CHECK TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, and `OPTIMIZE TABLE` in a convenient way for the user. It determines which statements to use for the operation you want to perform, and then sends the statements to the server to be executed. For details about which storage engines each statement works with, see the descriptions for those statements in [Chapter 12, SQL Statement Syntax](#).

The `MyISAM` storage engine supports all four statements, so `mysqlcheck` can be used to perform all four operations on `MyISAM` tables. Other storage engines do not necessarily support all operations. In such cases, an error message is displayed. For example, if `test.t` is a `MEMORY` table, an attempt to check it produces this result:

```
shell> mysqlcheck test t
test.t
note      : The storage engine for the table doesn't support check
```

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to filesystem errors.

There are three general ways to invoke `mysqlcheck`:

```
shell> mysqlcheck [options] db_name [tables]
shell> mysqlcheck [options] --databases db_name1 [db_name2 db_name3...]
shell> mysqlcheck [options] --all-databases
```

If you do not name any tables following `db_name` or if you use the `--databases` or `--all-databases` option, entire databases are checked.

`mysqlcheck` has a special feature compared to other client programs. The default behavior of checking tables (`--check`) can be changed by renaming the binary. If you want to have a tool that repairs tables by default, you should just make a copy of `mysqlcheck` named `mysqlrepair`, or make a symbolic link to `mysqlcheck` named `mysqlrepair`. If you invoke `mysqlrepair`, it repairs tables.

The following names can be used to change `mysqlcheck` default behavior:

<code>mysqlrepair</code>	The default option is <code>--repair</code>
<code>mysqlanalyze</code>	The default option is <code>--analyze</code>
<code>mysqloptimize</code>	The default option is <code>--optimize</code>

`mysqlcheck` supports the following options:

- `--help, -?`

Display a help message and exit.

- `--all-databases, -A`

Check all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line.

- `--all-in-1, -1`

Instead of issuing a statement for each table, execute a single statement for each database that names all the tables from that database to be processed.

- `--analyze, -a`

Analyze the tables.

MySQL Enterprise

For expert advice on optimizing tables, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `--auto-repair`

If a checked table is corrupted, automatically fix it. Any necessary repairs are done after all tables have been checked.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--check, -c`

Check the tables for errors. This is the default operation.

- `--check-only-changed, -C`

Check only tables that have changed since the last check or that have not been closed properly.

- `--check-upgrade, -g`

Invoke `CHECK TABLE` with the `FOR UPGRADE` option to check tables for incompatibilities with the current version of the server. This option automatically enables the `--fix-db-names` and `--fix-table-names` options. `--check-upgrade` was added in MySQL 5.1.7.

- `--compress`

Compress all information sent between the client and the server if both support compression.

- `--databases, -B`

Process all tables in the named databases. Normally, `mysqlcheck` treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. A typical `debug_options` string is often `'d:t:o,file_name'`.

- `--debug-check`

Print some debugging information when the program exits. This option was added in MySQL 5.1.21.

- `--debug-info`

Print debugging information and memory and CPU usage statistics when the program exits. This option was added in MySQL 5.1.21.

- `--default-character-set=charset_name`

Use `charset_name` as the default character set. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--extended, -e`

If you are using this option to check tables, it ensures that they are 100% consistent but takes a long time.

If you are using this option to repair tables, it runs an extended repair that may not only take a long time to execute, but may produce a lot of garbage rows also!

- `--fast, -F`

Check only tables that have not been closed properly.

- `--fix-db-names`

Convert database names to 5.1 format. Only database names that contain special characters are affected. This option was added in MySQL 5.1.7.

- `--fix-table-names`

Convert table names to 5.1 format. Only table names that contain special characters are affected. This option was added in MySQL 5.1.7.

- `--force, -f`

Continue even if an SQL error occurs.

- `--host=host_name, -h host_name`

Connect to the MySQL server on the given host.

- `--medium-check, -m`

Do a check that is faster than an `--extended` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--optimize, -o`

Optimize the tables.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--quick, -q`

If you are using this option to check tables, it prevents the check from scanning the rows to check for incorrect links. This is the fastest check method.

If you are using this option to repair tables, it tries to repair only the index tree. This is the fastest repair method.

- `--repair, -r`

Perform a repair that can fix almost anything except unique keys that are not unique.

- `--silent, -s`

Silent mode. Print only error messages.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.8.7.3, “SSL Command Options”](#).

- `--tables`

Overrides the `--databases` or `-B` option. All name arguments following the option are regarded as table names.

- `--use-frm`

For repair operations on `MyISAM` tables, get the table structure from the `.frm` file so that the table can be repaired even if the `.MYI` header is corrupted.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print information about the various stages of program operation.

- `--version, -V`

Display version information and exit.

7.13. `mysqldump` — A Database Backup Program

The `mysqldump` client is a backup program originally written by Igor Romanenko. It can be used to dump a database or a collection of databases for backup or transfer to another SQL server (not necessarily a MySQL server). The dump typically contains SQL statements to create the table, populate it, or both. However, `mysqldump` can also be used to generate files in CSV, other delimited text, or XML format.

If you are doing a backup on the server and your tables all are `MyISAM` tables, consider using the `mysqlhotcopy` instead because it can accomplish faster backups and faster restores. See [Section 7.14, “mysqlhotcopy — A Database Backup Program”](#).

There are three general ways to invoke `mysqldump`:

```
shell> mysqldump [options] db_name [tables]
shell> mysqldump [options] --databases db_name1 [db_name2 db_name3...]
shell> mysqldump [options] --all-databases
```

If you do not name any tables following `db_name` or if you use the `--databases` or `--all-databases` option, entire databases are dumped.

To get a list of the options your version of `mysqldump` supports, execute `mysqldump --help`.

Some `mysqldump` options are shorthand for groups of other options. `--opt` and `--compact` fall into this category. For example, use of `--opt` is the same as specifying `--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset`. Note that as of MySQL 5.1, all of the options that `--opt` stands for also are on by default because `--opt` is on by default.

To reverse the effect of a group option, uses its `--skip-xxx` form (`--skip-opt` or `--skip-compact`). It is also possible to select only part of the effect of a group option by following it with options that enable or disable specific features. Here are some examples:

- To select the effect of `--opt` except for some features, use the `--skip` option for each feature. For example, to disable extended

inserts and memory buffering, use `--opt --skip-extended-insert --skip-quick`. (As of MySQL 5.1, `--skip-extended-insert --skip-quick` is sufficient because `--opt` is on by default.)

- To reverse `--opt` for all features except index disabling and table locking, use `--skip-opt --disable-keys --lock-tables`.

When you selectively enable or disable the effect of a group option, order is important because options are processed first to last. For example, `--disable-keys --lock-tables --skip-opt` would not have the intended effect; it is the same as `--skip-opt` by itself.

`mysqldump` can retrieve and dump table contents row by row, or it can retrieve the entire content from a table and buffer it in memory before dumping it. Buffering in memory can be a problem if you are dumping large tables. To dump tables row by row, use the `--quick` option (or `--opt`, which enables `--quick`). `--opt` (and hence `--quick`) is enabled by default as of MySQL 5.1 to enable memory buffering, use `--skip-quick`.

If you are using a recent version of `mysqldump` to generate a dump to be reloaded into a very old MySQL server, you should not use the `--opt` or `--extended-insert` option. Use `--skip-opt` instead.

Before MySQL 4.1.2, out-of-range numeric values such as `-inf` and `inf`, as well as `NaN` (not-a-number) values are dumped by `mysqldump` as `NULL`. You can see this using the following sample table:

```
mysql> CREATE TABLE t (f DOUBLE);
mysql> INSERT INTO t VALUES(1e+1111111111111111111);
mysql> INSERT INTO t VALUES(-1e1111111111111111111);
mysql> SELECT f FROM t;
+-----+
| f      |
+-----+
| inf    |
| -inf   |
+-----+
```

For this table, `mysqldump` produces the following data output:

```
--
-- Dumping data for table `t`
--
INSERT INTO t VALUES (NULL);
INSERT INTO t VALUES (NULL);
```

The significance of this behavior is that if you dump and restore the table, the new table has contents that differ from the original contents. This problem is fixed as of MySQL 4.1.2; you cannot insert `inf` in the table, so this `mysqldump` behavior is only relevant when you deal with old servers.

Note

`mysqldump` from the MySQL 5.1.21 distribution cannot be used to create dumps from MySQL server versions 5.1.20 and older. This issue is fixed in MySQL 5.1.22. ([Bug#30123](#))

`mysqldump` supports the following options:

- `--help, -?`

Display a help message and exit.

- `--add-drop-database`

Add a `DROP DATABASE` statement before each `CREATE DATABASE` statement.

- `--add-drop-table`

Add a `DROP TABLE` statement before each `CREATE TABLE` statement.

- `--add-locks`

Surround each table dump with `LOCK TABLES` and `UNLOCK TABLES` statements. This results in faster inserts when the dump file

is reloaded. See [Section 6.2.17, “Speed of INSERT Statements”](#).

- **--all-databases, -A**

Dump all tables in all databases. This is the same as using the **--databases** option and naming all the databases on the command line.

- **--all-tablespaces, -Y**

Adds to a table dump all SQL statements needed to create any tablespaces used by an **NDB Cluster** table. This information is not otherwise included in the output from **mysqldump**. This option is currently relevant only to MySQL Cluster tables.

This option was added in MySQL 5.1.6.

- **--allow-keywords**

Allow creation of column names that are keywords. This works by prefixing each column name with the table name.

- **--character-sets-dir=path**

The directory where character sets are installed. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- **--comments, -i**

Write additional information in the dump file such as program version, server version, and host. This option is enabled by default. To suppress this additional information, use **--skip-comments**.

- **--compact**

Produce less verbose output. This option suppresses comments and enables the **--skip-add-drop-table**, **--skip-set-charset**, **--skip-disable-keys**, and **--skip-add-locks** options.

Note

Prior to release 5.1.21, this option did not create valid SQL if the database dump contained views. The recreation of views requires the creation and removal of temporary tables and this option suppressed the removal of those temporary tables. As a workaround use **--compress** with the **--add-drop-table** option and then manually adjust the dump file.

- **--compatible=name**

Produce output that is more compatible with other database systems or with older MySQL servers. The value of **name** can be **ansi**, **mysql323**, **mysql40**, **postgresql**, **oracle**, **mssql**, **db2**, **maxdb**, **no_key_options**, **no_table_options**, or **no_field_options**. To use several values, separate them by commas. These values have the same meaning as the corresponding options for setting the server SQL mode. See [Section 5.2.6, “SQL Modes”](#).

This option does not guarantee compatibility with other servers. It only enables those SQL mode values that are currently available for making dump output more compatible. For example, **--compatible=oracle** does not map data types to Oracle types or use Oracle comment syntax.

This option requires a server version of 4.1.0 or higher. With older servers, it does nothing.

- **--complete-insert, -c**

Use complete **INSERT** statements that include column names.

- **--compress, -C**

Compress all information sent between the client and the server if both support compression.

- **--create-options**

Include all MySQL-specific table options in the **CREATE TABLE** statements.

- **--databases, -B**

Dump several databases. Normally, **mysqldump** treats the first name argument on the command line as a database name and fol-

lowering names as table names. With this option, it treats all name arguments as database names. `CREATE DATABASE` and `USE` statements are included in the output before each new database.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The `debug_options` string is often `'d:t:o,file_name'`. The default value is `'d:t:o,/tmp/mysqldump.trace'`.

- `--debug-check`

Print some debugging information when the program exits. This option was added in MySQL 5.1.21.

- `--debug-info`

Print debugging information and memory and CPU usage statistics when the program exits. This option was added in MySQL 5.1.21.

- `--default-character-set=charset_name`

Use `charset_name` as the default character set. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#). If no character set is specified, `mysqldump` uses `utf8`, and earlier versions use `latin1`.

- `--delayed-insert`

Write `INSERT DELAYED` statements rather than `INSERT` statements.

- `--delete-master-logs`

On a master replication server, delete the binary logs after performing the dump operation. This option automatically enables `--master-data`.

- `--disable-keys, -K`

For each table, surround the `INSERT` statements with `/*!40000 ALTER TABLE tbl_name DISABLE KEYS */;` and `/*!40000 ALTER TABLE tbl_name ENABLE KEYS */;` statements. This makes loading the dump file faster because the indexes are created after all rows are inserted. This option is effective only for non-unique indexes of `MyISAM` tables.

- `--events, -E`

Dump events from the dumped databases. This option was added in MySQL 5.1.8.

- `--extended-insert, -e`

Use multiple-row `INSERT` syntax that include several `VALUES` lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.

- `--fields-terminated-by=..., --fields-enclosed-by=..., --fields-optionally-enclosed-by=..., --fields-escaped-by=...`

These options are used with the `-T` option and have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. See [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).

- `--first-slave, -x`

Deprecated. Now renamed to `--lock-all-tables`.

- `--flush-logs, -F`

Flush the MySQL server log files before starting the dump. This option requires the `RELOAD` privilege. Note that if you use this option in combination with the `--all-databases` (or `-A`) option, the logs are flushed *for each database dumped*. The exception is when using `--lock-all-tables` or `--master-data`: In this case, the logs are flushed only once, corresponding to the moment that all tables are locked. If you want your dump and the log flush to happen at exactly the same moment, you should use `--flush-logs` together with either `--lock-all-tables` or `--master-data`.

- `--flush-privileges`

Emit a `FLUSH PRIVILEGES` statement after dumping the `mysql` database. This option should be used any time the dump contains the `mysql` database and any other database that depends on the data in the `mysql` database for proper restoration. This option was added in MySQL 5.1.12.

- `--force, -f`

Continue even if an SQL error occurs during a table dump.

One use for this option is to cause `mysqldump` to continue executing even when it encounters a view that has become invalid because the definition refers to a table that has been dropped. Without `--force`, `mysqldump` exits with an error message. With `--force`, `mysqldump` prints the error message, but it also writes a SQL comment containing the view definition to the dump output and continues executing.

- `--host=host_name, -h host_name`

Dump data from the MySQL server on the given host. The default host is `localhost`.

- `--hex-blob`

Dump binary columns using hexadecimal notation (for example, `'abc'` becomes `0x616263`). The affected data types are `BINARY`, `VARBINARY`, `BLOB`, and `BIT`.

- `--ignore-table=db_name.tbl_name`

Do not dump the given table, which must be specified using both the database and table names. To ignore multiple tables, use this option multiple times.

- `--insert-ignore`

Write `INSERT` statements with the `IGNORE` option.

- `--lines-terminated-by=...`

This option is used with the `-T` option and has the same meaning as the corresponding clause for `LOAD DATA INFILE`. See [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).

- `--lock-all-tables, -x`

Lock all tables across all databases. This is achieved by acquiring a global read lock for the duration of the whole dump. This option automatically turns off `--single-transaction` and `--lock-tables`.

- `--lock-tables, -l`

Lock all tables before dumping them. The tables are locked with `READ LOCAL` to allow concurrent inserts in the case of `MyISAM` tables. For transactional tables such as `InnoDB` and `BDB`, `--single-transaction` is a much better option, because it does not need to lock the tables at all.

Please note that when dumping multiple databases, `--lock-tables` locks tables for each database separately. Therefore, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states.

- `--master-data[=value]`

Write the binary log filename and position to the output. This option requires the `RELOAD` privilege and the binary log must be enabled. If the option value is equal to 1, the position and filename are written to the dump output in the form of a `CHANGE MASTER` statement. If the dump is from a master server and you use it to set up a slave server, the `CHANGE MASTER` statement causes the slave to start from the correct position in the master's binary logs. If the option value is equal to 2, the `CHANGE MASTER` statement is written as an SQL comment. If the value is not specified, then the default value is 1.

The `--master-data` option automatically turns off `--lock-tables`. It also turns on `--lock-all-tables`, unless `--single-transaction` also is specified (in which case, a global read lock is acquired only for a short time at the beginning of the dump. See also the description for `--single-transaction`. In all cases, any action on logs happens at the exact moment of the dump.

- `--no-autocommit`

Enclose the `INSERT` statements for each dumped table within `SET AUTOCOMMIT=0` and `COMMIT` statements.

- `--no-create-db, -n`

This option suppresses the `CREATE DATABASE` statements that are otherwise included in the output if the `--databases` or `--all-databases` option is given.

- `--no-create-info, -t`

Do not write `CREATE TABLE` statements that re-create each dumped table.

- `--no-data, -d`

Do not write any table row information (that is, do not dump table contents). This is very useful if you want to dump only the `CREATE TABLE` statement for the table.

- `--opt`

This option is shorthand; it is the same as specifying `--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset`. It should give you a fast dump operation and produce a dump file that can be reloaded into a MySQL server quickly.

The `--opt` option is enabled by default. Use `--skip-opt` to disable it. See the discussion at the beginning of this section for information about selectively enabling or disabling certain of the options affected by `--opt`.

- `--order-by-primary`

Sorts each table's rows by its primary key, or by its first unique index, if such an index exists. This is useful when dumping a `MyISAM` table to be loaded into an `InnoDB` table, but will make the dump itself take considerably longer.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--quick, -q`

This option is useful for dumping large tables. It forces `mysqldump` to retrieve rows for a table from the server a row at a time rather than retrieving the entire row set and buffering it in memory before writing it out.

- `--quote-names, -Q`

Quote database, table, and column names within ‘`’ characters. If the `ANSI_QUOTES` SQL mode is enabled, names are quoted within ‘”’ characters. This option is enabled by default. It can be disabled with `--skip-quote-names`, but this option should be given after any option such as `--compatible` that may enable `--quote-names`.

- `--replace`

Write `REPLACE` statements rather than `INSERT` statements. Available as of MySQL 5.1.3.

- `--result-file=file, -r file`

Direct output to a given file. This option should be used on Windows to prevent newline ‘\n’ characters from being converted to ‘\r\n’ carriage return/newline sequences. The result file is created and its contents overwritten, even if an error occurs while gen-

erating the dump. The previous contents are lost.

- `--routines, -R`

Dump stored routines (functions and procedures) from the dumped databases. Use of this option requires the `SELECT` privilege for the `mysql.proc` table. The output generated by using `--routines` contains `CREATE PROCEDURE` and `CREATE FUNCTION` statements to re-create the routines. However, these statements do not include attributes such as the routine creation and modification timestamps. This means that when the routines are reloaded, they will be created with the timestamps equal to the reload time.

If you require routines to be re-created with their original timestamp attributes, do not use `--routines`. Instead, dump and reload the contents of the `mysql.proc` table directly, using a MySQL account that has appropriate privileges for the `mysql` database.

This option was added in MySQL 5.1.2. Before that, stored routines are not dumped. Routine `DEFINER` values are not dumped until MySQL 5.1.8. This means that before 5.1.8, when routines are reloaded, they will be created with the definer set to the reloading user. If you require routines to be re-created with their original definer, dump and load the contents of the `mysql.proc` table directly as described earlier.

- `--set-charset`

Add `SET NAMES default_character_set` to the output. This option is enabled by default. To suppress the `SET NAMES` statement, use `--skip-set-charset`.

- `--single-transaction`

This option issues a `BEGIN` SQL statement before dumping data from the server. It is useful only with transactional tables such as `InnoDB`, because then it dumps the consistent state of the database at the time when `BEGIN` was issued without blocking any applications.

When using this option, you should keep in mind that only `InnoDB` tables are dumped in a consistent state. For example, any `MyISAM` or `MEMORY` tables dumped while using this option may still change state.

While a `--single-transaction` dump is in process, to ensure a valid dump file (correct table contents and binary log position), no other connection should use the following statements: `ALTER TABLE`, `DROP TABLE`, `RENAME TABLE`, `TRUNCATE TABLE`. A consistent read is not isolated from those statements, so use of them on a table to be dumped can cause the `SELECT` performed by `mysqldump` to retrieve the table contents to obtain incorrect contents or fail.

This option is not supported for MySQL Cluster tables; the results cannot be guaranteed to be consistent due to the fact that the `NDB-Cluster` storage engine supports only the `READ_COMMITTED` transaction isolation level. You should always use `NDB` backup and restore instead.

The `--single-transaction` option and the `--lock-tables` option are mutually exclusive, because `LOCK TABLES` causes any pending transactions to be committed implicitly.

To dump large tables, you should combine this option with `--quick`.

- `--skip-comments`

See the description for the `--comments` option.

- `--skip-opt`

See the description for the `--opt` option.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.8.7.3, “SSL Command Options”](#).

- `--tab=path, -T path`

Produce tab-separated data files. For each dumped table, `mysqldump` creates a `tbl_name.sql` file that contains the `CREATE TABLE` statement that creates the table, and a `tbl_name.txt` file that contains its data. The option value is the directory in which

to write the files.

By default, the `.txt` data files are formatted using tab characters between column values and a newline at the end of each line. The format can be specified explicitly using the `--fields-xxx` and `--lines-terminated-by` options.

Note

This option should be used only when `mysqldump` is run on the same machine as the `mysqld` server. You must have the `FILE` privilege, and the server must have permission to write files in the directory that you specify.

- `--tables`

Override the `--databases` or `-B` option. `mysqldump` regards all name arguments following the option as table names.

- `--triggers`

Dump triggers for each dumped table. This option is enabled by default; disable it with `--skip-triggers`.

- `--tz-utc`

Add `SET TIME_ZONE='+00:00'` to the dump file so that `TIMESTAMP` columns can be dumped and reloaded between servers in different time zones. Without this option, `TIMESTAMP` columns are dumped and reloaded in the time zones local to the source and destination servers, which can cause the values to change. `--tz-utc` also protects against changes due to daylight saving time. `--tz-utc` is enabled by default. To disable it, use `--skip-tz-utc`. This option was added in MySQL 5.1.2.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print more information about what the program does.

- `--version, -V`

Display version information and exit.

- `--where='where_condition', -w 'where_condition'`

Dump only rows selected by the given `WHERE` condition. Quotes around the condition are mandatory if it contains spaces or other characters that are special to your command interpreter.

Examples:

```
--where="user='jimf'"
-w"userid>1"
-w"userid<1"
```

- `--xml, -X`

Write dump output as well-formed XML.

NULL, 'NULL', and Empty Values: For some column named `column_name`, the `NULL` value, an empty string, and the string value `'NULL'` are distinguished from one another in the output generated by this option as follows:

Value:	XML Representation:
<code>NULL</code> (unknown value)	<code><field name="column_name" xsi:nil="true" /></code>
<code>' '</code> (empty string)	<code><field name="column_name"></field></code>
<code>'NULL'</code> (string value)	<code><field name="column_name">NULL</field></code>

Beginning with MySQL 5.1.12, the output from the `mysql` client when run using the `--xml` option also follows these rules. (See [Section 7.8.1, “mysql Options”](#).)

Beginning with MySQL 5.1.18, XML output from `mysqldump` includes the XML namespace, as shown here:

```
shell> mysqldump --xml -u root world City
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="world">
<table_structure name="City">
<field Field="ID" Type="int(11)" Null="NO" Key="PRI" Extra="auto_increment" />
<field Field="Name" Type="char(35)" Null="NO" Key="" Default="" Extra="" />
<field Field="CountryCode" Type="char(3)" Null="NO" Key="" Default="" Extra="" />
<field Field="District" Type="char(20)" Null="NO" Key="" Default="" Extra="" />
<field Field="Population" Type="int(11)" Null="NO" Key="" Default="0" Extra="" />
<key Table="City" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="ID" Collation="A" Cardinality="4079"
Null="" Index_type="BTREE" Comment="" />
<options Name="City" Engine="MyISAM" Version="10" Row_format="Fixed" Rows="4079" Avg_row_length="67" Data_length="27329
3" Max_data_length="18858823439613951" Index_length="43008" Data_free="0" Auto_increment="4080" Create_time="2007-03-31
e_time="2007-03-31 01:47:02" Collation="latin1_swedish_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="City">
<row>
<field name="ID">1</field>
<field name="Name">Kabul</field>
<field name="CountryCode">AFG</field>
<field name="District">Kabul</field>
<field name="Population">1780000</field>
</row>
...
<row>
<field name="ID">4079</field>
<field name="Name">Rafah</field>
<field name="CountryCode">PSE</field>
<field name="District">Rafah</field>
<field name="Population">92020</field>
</row>
</table_data>
</database>
</mysqldump>
```

You can also set the following variables by using `--var_name=value` syntax:

- `max_allowed_packet`

The maximum size of the buffer for client/server communication. The maximum is 1GB.

- `net_buffer_length`

The initial size of the buffer for client/server communication. When creating multiple-row-insert statements (as with option `--extended-insert` or `--opt`), `mysqldump` creates rows up to `net_buffer_length` length. If you increase this variable, you should also ensure that the `net_buffer_length` variable in the MySQL server is at least this large.

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated.*

The most common use of `mysqldump` is probably for making a backup of an entire database:

```
shell> mysqldump db_name > backup-file.sql
```

You can read the dump file back into the server like this:

```
shell> mysql db_name < backup-file.sql
```

Or like this:

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```

`mysqldump` is also very useful for populating databases by copying data from one MySQL server to another:

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

It is possible to dump several databases with one command:

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.sql
```

To dump all databases, use the `--all-databases` option:

```
shell> mysqldump --all-databases > all_databases.sql
```

For **InnoDB** tables, `mysqldump` provides a way of making an online backup:

```
shell> mysqldump --all-databases --single-transaction > all_databases.sql
```

This backup just needs to acquire a global read lock on all tables (using `FLUSH TABLES WITH READ LOCK`) at the beginning of the dump. As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If and only if one long updating statement is running when the `FLUSH` statement is issued, the MySQL server may get stalled until that long statement finishes, and then the dump becomes lock-free. If the update statements that the MySQL server receives are short (in terms of execution time), the initial lock period should not be noticeable, even with many updates.

For point-in-time recovery (also known as “roll-forward,” when you need to restore an old backup and replay the changes that happened since that backup), it is often useful to rotate the binary log (see [Section 5.11.4, “The Binary Log”](#)) or at least know the binary log coordinates to which the dump corresponds:

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

Or:

```
shell> mysqldump --all-databases --flush-logs --master-data=2  
> all_databases.sql
```

The `--master-data` and `--single-transaction` options can be used simultaneously, which provides a convenient way to make an online backup suitable for point-in-time recovery if tables are stored using the **InnoDB** storage engine.

For more information on making backups, see [Section 5.9.1, “Database Backups”](#), and [Section 5.9.2, “Example Backup and Recovery Strategy”](#).

If you encounter problems backing up views, please read the section that covers restrictions on views which describes a workaround for backing up views when this fails due to insufficient privileges. See [Section D.4, “Restrictions on Views”](#).

MySQL Enterprise

MySQL Enterprise subscribers will find more information about `mysqldump` in the Knowledge Base article, [How Can I Avoid Inserting Duplicate Rows From a Dump File?](#). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

7.14. `mysqlhotcopy` — A Database Backup Program

`mysqlhotcopy` is a Perl script that was originally written and contributed by Tim Bunce. It uses `LOCK TABLES`, `FLUSH TABLES`, and `cp` or `scp` to make a database backup quickly. It is the fastest way to make a backup of the database or single tables, but it can be run only on the same machine where the database directories are located. `mysqlhotcopy` works only for backing up **MyISAM** and **ARCHIVE** tables. It runs on Unix and NetWare.

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
```

```
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
```

Back up tables in the given database that match a regular expression:

```
shell> mysqlhotcopy db_name ./regex/
```

The regular expression for the table name can be negated by prefixing it with a tilde (`~`):

```
shell> mysqlhotcopy db_name ./~regex/
```

`mysqlhotcopy` supports the following options:

- `--help, -?`
Display a help message and exit.
- `--addtodest`
Do not rename target directory (if it exists); merely add files to it.
- `--allowold`
Do not abort if a target exists; rename it by adding an `_old` suffix.
- `--checkpoint=db_name.tbl_name`
Insert checkpoint entries into the specified database `db_name` and table `tbl_name`.
- `--chroot=path`
Base directory of the `chroot` jail in which `mysqld` operates. The `path` value should match that of the `--chroot` option given to `mysqld`.
- `--debug`
Enable debug output.
- `--dryrun, -n`
Report actions without performing them.
- `--flushlog`
Flush logs after all tables are locked.
- `--host=host_name, -h host_name`
The hostname of the local host to use for making a TCP/IP connection to the local server. By default, the connection is made to `localhost` using a Unix socket file.
- `--keepold`
Do not delete previous (renamed) target when done.
- `--method=command`
The method for copying files (`cp` or `scp`).
- `--noindices`
Do not include full index files in the backup. This makes the backup smaller and faster. The indexes for reloaded tables can be re-constructed later with `myisamchk -rq`.
- `--password=password, -ppassword`
The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs. You can use an option file to avoid giving the password on the command line.
Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).
- `--port=port_num, -P port_num`
The TCP/IP port number to use when connecting to the local server.
- `--quiet, -q`

Be silent except for errors.

- `--record_log_pos=db_name.tbl_name`

Record master and slave status in the specified database `db_name` and table `tbl_name`.

- `--regex=expr`

Copy all databases with names that match the given regular expression.

- `--resetmaster`

Reset the binary log after locking all the tables.

- `--resetslave`

Reset the `master.info` file after locking all the tables.

- `--socket=path, -S path`

The Unix socket file to use for the connection.

- `--suffix=str`

The suffix for names of copied databases.

- `--tmpdir=path`

The temporary directory. The default is `/tmp`.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

`mysqlhotcopy` reads the `[client]` and `[mysqlhotcopy]` option groups from option files.

To execute `mysqlhotcopy`, you must have access to the files for the tables that you are backing up, the `SELECT` privilege for those tables, the `RELOAD` privilege (to be able to execute `FLUSH TABLES`), and the `LOCK TABLES` privilege (to be able to lock the tables).

Use `perldoc` for additional `mysqlhotcopy` documentation, including information about the structure of the tables needed for the `-checkpoint` and `--record_log_pos` options:

```
shell> perldoc mysqlhotcopy
```

MySQL Enterprise

MySQL Enterprise subscribers will find more information about `mysqlhotcopy` in the Knowledge Base article, [How Does mysqlhotcopy Work?](http://www.mysql.com/products/enterprise/advisors.html). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

7.15. `mysqlimport` — A Data Import Program

The `mysqlimport` client provides a command-line interface to the `LOAD DATA INFILE` SQL statement. Most options to `mysqlimport` correspond directly to clauses of `LOAD DATA INFILE` syntax. See [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).

Invoke `mysqlimport` like this:

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

For each text file named on the command line, `mysqlimport` strips any extension from the filename and uses the result to determine the name of the table into which to import the file's contents. For example, files named `patient.txt`, `patient.text`, and `patient` all would be imported into a table named `patient`.

`mysqlimport` supports the following options:

- `--help, -?`
Display a help message and exit.
- `--character-sets-dir=path`
The directory where character sets are installed. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).
- `--columns=column_list, -c column_list`
This option takes a comma-separated list of column names as its value. The order of the column names indicates how to match data file columns with table columns.
- `--compress, -C`
Compress all information sent between the client and the server if both support compression.
- `--debug[=debug_options], -# [debug_options]`
Write a debugging log. The `debug_options` string often is `'d:t:o,file_name'`.
- `--debug-check`
Print some debugging information when the program exits. This option was added in MySQL 5.1.21.
- `--debug-info`
Print debugging information and memory and CPU usage statistics when the program exits. This option was added in MySQL 5.1.21.
- `--default-character-set=charset_name`
Use `charset_name` as the default character set. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).
- `--delete, -D`
Empty the table before importing the text file.
- `--fields-terminated-by=..., --fields-enclosed-by=..., --fields-optionally-enclosed-by=..., --fields-escaped-by=...`
These options have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. See [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).
- `--force, -f`
Ignore errors. For example, if a table for a text file does not exist, continue processing any remaining files. Without `--force`, `mysqlimport` exits if a table does not exist.
- `--host=host_name, -h host_name`
Import data to the MySQL server on the given host. The default host is `localhost`.
- `--ignore, -i`
See the description for the `--replace` option.
- `--ignore-lines=N`
Ignore the first `N` lines of the data file.
- `--lines-terminated-by=...`
This option has the same meaning as the corresponding clause for `LOAD DATA INFILE`. For example, to import Windows files

that have lines terminated with carriage return/linefeed pairs, use `--lines-terminated-by="\r\n"`. (You might have to double the backslashes, depending on the escaping conventions of your command interpreter.) See [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).

- `--local, -L`

Read input files locally from the client host.

MySQL Enterprise

For expert advice on the security implications of enabling LOCAL, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `--lock-tables, -l`

Lock *all* tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

- `--low-priority`

Use `LOW_PRIORITY` when loading the table. This affects only storage engines that use only table-level locking (`MyISAM`, `MEMORY`, `MERGE`).

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--replace, -r`

The `--replace` and `--ignore` options control handling of input rows that duplicate existing rows on unique key values. If you specify `--replace`, new rows replace existing rows that have the same unique key value. If you specify `--ignore`, input rows that duplicate an existing row on a unique key value are skipped. If you do not specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

- `--silent, -s`

Silent mode. Produce output only when errors occur.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.8.7.3, “SSL Command Options”](#).

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print more information about what the program does.

- `--version, -V`

Display version information and exit.

Here is a sample session that demonstrates use of `mysqlimport`:

```
shell> mysql -e 'CREATE TABLE impptest(id INT, n VARCHAR(30))' test
shell> ed
a
100      Max Sydow
101      Count Dracula
.
w impptest.txt
32
q
shell> od -c impptest.txt
00000000  1  0  0  \t  M  a  x           S  y  d  o  w  \n  1  0
00000020  1  \t  C  o  u  n  t           D  r  a  c  u  l  a  \n
00000040
shell> mysqlimport --local test impptest.txt
test. impptest: Records: 2 Deleted: 0 Skipped: 0 Warnings: 0
shell> mysql -e 'SELECT * FROM impptest' test
+-----+-----+
| id | n |
+-----+-----+
| 100 | Max Sydow |
| 101 | Count Dracula |
+-----+-----+
```

7.16. `mysqlshow` — Display Database, Table, and Column Information

The `mysqlshow` client can be used to quickly see which databases exist, their tables, or a table's columns or indexes.

`mysqlshow` provides a command-line interface to several SQL `SHOW` statements. See [Section 12.5.4, “SHOW Syntax”](#). The same information can be obtained by using those statements directly. For example, you can issue them from the `mysql` client program.

Invoke `mysqlshow` like this:

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- If no database is given, a list of database names is shown.
- If no table is given, all matching tables in the database are shown.
- If no column is given, all matching columns and column types in the table are shown.

The output displays only the names of those databases, tables, or columns for which you have some privileges.

If the last argument contains shell or SQL wildcard characters (`*`, `?`, `%`, or `_`), only those names that are matched by the wildcard are shown. If a database name contains any underscores, those should be escaped with a backslash (some Unix shells require two) to get a list of the proper tables or columns. `*` and `?` characters are converted into SQL `%` and `_` wildcard characters. This might cause some confusion when you try to display the columns for a table with a `_` in the name, because in this case, `mysqlshow` shows you only the table names that match the pattern. This is easily fixed by adding an extra `%` last on the command line as a separate argument.

`mysqlshow` supports the following options:

- `--help, -?`

Display a help message and exit.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--compress, -C`

Compress all information sent between the client and the server if both support compression.

- `--count`

Show the number of rows per table. This can be slow for non-MyISAM tables.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The `debug_options` string often is `'d:t:o,file_name'`.

- `--debug-check`

Print some debugging information when the program exits. This option was added in MySQL 5.1.21.

- `--debug-info`

Print debugging information and memory and CPU usage statistics when the program exits. This option was added in MySQL 5.1.21.

- `--default-character-set=charset_name`

Use `charset_name` as the default character set. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- `--host=host_name, -h host_name`

Connect to the MySQL server on the given host.

- `--keys, -k`

Show table indexes.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--show-table-type, -t`

Show a column indicating the table type, as in `SHOW FULL TABLES`. The type is `BASE TABLE` or `VIEW`.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.8.7.3, “SSL Command Options”](#).

- `--status, -i`

Display extra information about each table.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version, -V`

Display version information and exit.

7.17. `mysqlslap` — Load Emulation Client

`mysqlslap` is a diagnostic program designed to emulate client load for a MySQL server and to report the timing of each stage. It works as if multiple clients are accessing the server. `mysqlslap` is available as of MySQL 5.1.4.

Invoke `mysqlslap` like this:

```
shell> mysqlslap [options]db_name
```

Some options such as `--create` or `--query` enable you to specify a string containing an SQL statement or a file containing statements. If you specify a file, by default it must contain one statement per line. (That is, the implicit statement delimiter is the newline character.) Use the `--delimiter` to specify different delimiter, which allows you to specify statements that span multiple lines or place multiple statements on a single line. You cannot include comments in a file; `mysqlslap` does not understand them.

`mysqlslap` supports the following options:

- `--help, -?`

Display a help message and exit.

- `--auto-generate-sql, -a`

Generate SQL statements automatically when they are not supplied in files or via command options.

- `--compress, -C`

Compress all information sent between the client and the server if both support compression.

- `--concurrency=N, -c N`

The number of clients to simulate when issuing the `SELECT` statement.

- `--create=value`

The file or string to use for creating the table.

- `--create-schema=value`

The schema in which to run the tests. This option was added in MySQL 5.1.5.

- `--csv[=file]`

Generate output in comma-separated values format. The output goes to the named file, or to the standard output if no file is given. This option was added in MySQL 5.1.5.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The `debug_options` string often is `'d:t:o,file_name'`.

- `--debug-check`

Print some debugging information when the program exits. This option was added in MySQL 5.1.21.

- `--debug-info, -T`

Print debugging information and memory and CPU usage statistics when the program exits. This option was added in MySQL 5.1.21.

- `--delimiter=str, -F str`

The delimiter to use in SQL statements supplied in files or via command options.

- `--engine=engine_name, -e engine_name`

The storage engine to use for creating the table.

- `--host=host_name, -h host_name`

Connect to the MySQL server on the given host.

- `--iterations=N, -i N`

The number of times to run the tests.

- `--lock-directory=path`

The directory to use for storing locks. This option was added in MySQL 5.1.5.

- `--number-char-cols=N, -x N`

The number of `VARCHAR` columns to use if `--auto-generate-sql` is specified.

- `--number-int-cols=N, -y N`

The number of `INT` columns to use if `--auto-generate-sql` is specified.

- `--number-of-queries=N`

Limit each client to approximately this number of queries. This option was added in MySQL 5.1.5.

- `--only-print`

Do not connect to databases. `mysqlslap` only prints what it would have done. This option was added in MySQL 5.1.5.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the `password` value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--preserve-schema`

Preserve the schema from the `mysqlslap` run. This option was added in MySQL 5.1.5.

- `--query=value, -q value`

The file or string containing the `SELECT` statement to use for retrieving data.

- `--silent, -s`

Silent mode. No output.

- `--skip-query, -Q`

Don't run any `SELECT` statements.

- `--slave`

Follow master locks for other `mysqslap` clients. Use this option if you are trying to synchronize around one master server with `--lock-directory` plus NFS. This option was added in MySQL 5.1.5.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.8.7.3, “SSL Command Options”](#).

- `--use-threads`

On Unix, the default is to use `fork()` calls. This option causes `pthread` calls to be used instead. On Windows, the default is to use `pthread` calls and the option has no effect. This option was added in MySQL 5.1.6.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print more information about what the program does.

- `--version, -V`

Display version information and exit.

7.18. `mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine

`mysql_convert_table_format` converts the tables in a database to use a particular storage engine (`MyISAM` by default). `mysql_convert_table_format` is written in Perl and requires that the `DBI` and `DBD::mysql` Perl modules be installed (see [Section 2.15, “Perl Installation Notes”](#)).

The results may assist you in determining which queries result in table scans and where it would be beneficial to add indexes to your tables.

Invoke `mysql_convert_table_format` like this:

```
shell> mysql_convert_table_format [options]db_name
```

The `db_name` argument indicates the database containing the tables to be converted.

`mysql_convert_table_format` understands the options described in the following list.

- `--help, -?`

Display a help message and exit.

- `--force`

Continue even if errors occur.

- `--host=host_name, -h host_name`

Connect to the MySQL server on the given host.

- `--password=password, -p password`

The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs. You can use an option file to avoid giving the password on the command line.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--port=port_num`

The TCP/IP port number to use for the connection.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use.

- `--type=engine_name`

Specify the storage engine that the tables should be converted to use. The default is `MyISAM` if this option is not given.

MySQL Enterprise

For expert advice on choosing the optimum storage engine, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose`

Verbose mode. Print more information about what the program does.

- `--version`

Display version information and exit.

7.19. `mysql_find_rows` — Extract SQL Statements from Files

`mysql_find_rows` reads files containing SQL statements and extracts statements that match a given regular expression or that contain `USE db_name` or `SET` statements. The utility was written for use with update log files, but it can be used with other files that contain SQL statements.

Invoke `mysql_find_rows` like this:

```
shell> mysql_find_rows [options] [file_name ...]
```

Each `file_name` argument should be the name of file containing SQL statements. If no filenames are given, `mysql_find_rows` reads the standard input.

Examples:

```
mysql_find_rows --regexp=problem_table --rows=20 < update.log
mysql_find_rows --regexp=problem_table update-log.1 update-log.2
```

`mysql_find_rows` supports the following options:

- `--help, --Information`

Display a help message and exit.

- `--regexp=pattern`

Display queries that match the pattern.

- `--rows=N`

Quit after displaying *N* queries.

- `--skip-use-db`

Do not include `USE db_name` statements in the output.

- `--start_row=N`

Start output from this row.

7.20. `mysql_fix_extensions` — Make Table Filename Extensions Lowercase

`mysql_fix_extensions` converts the extensions for `MyISAM` table files to lowercase. It looks for files with an extension that matches any lettercase variant of `.frm`, `.myd`, `.myi`, `.isd`, and `.ism` and renames them to have extensions of `.frm`, `.MYD`, `.MYI`, `.ISD`, and `.ISM`, respectively. This can be useful after transferring the files from a system with case-insensitive filenames (such as Windows) to a system with case-sensitive filenames.

Invoke `mysql_fix_extensions` like this, where `data_dir` is the pathname to the MySQL data directory.

```
shell> mysql_fix_extensions data_dir
```

7.21. `mysql_setpermission` — Interactively Set Permissions in Grant Tables

`mysql_setpermission` is a Perl script that was originally written and contributed by Luuk de Boer. It interactively sets permissions in the MySQL grant tables. `mysql_setpermission` is written in Perl and requires that the `DBI` and `DBD:mysql` Perl modules be installed (see [Section 2.15](#), “Perl Installation Notes”).

Invoke `mysql_setpermission` like this:

```
shell> mysql_setpermission [options]
```

options should be either `--help` to display the help message, or options that indicate how to connect to the MySQL server. The account used when you connect determines which permissions you have when attempting to modify existing permissions in the grant tables.

`mysql_setpermissions` also reads options from the `[client]` and `[perl]` groups in the `.my.cnf` file in your home directory, if the file exists.

`mysql_setpermission` understands the following options:

- `--help`

Display a help message and exit.

- `--host=host_name`

Connect to the MySQL server on the given host.

- `--password=password`

The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs. You can use an option file to avoid giving the password on the command line.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--port=port_num`

The TCP/IP port number to use for the connection.

- `--socket=path`

For connections to `localhost`, the Unix socket file to use.

- `--user=user_name`

The MySQL username to use when connecting to the server.

7.22. `mysql_tableinfo` — Generate Database Metadata

`mysql_tableinfo` creates tables and populates them with database metadata. It uses `SHOW DATABASES`, `SHOW TABLES`, `SHOW TABLE STATUS`, `SHOW COLUMNS`, and `SHOW INDEX` to obtain the metadata.

In MySQL 5.0 and up, the `INFORMATION_SCHEMA` database contains the same kind of information in the `SCHEMATA`, `TABLES`, `COLUMNS`, and `STATISTICS` tables. See [Chapter 23, The `INFORMATION\_SCHEMA` Database](#).

Invoke `mysql_tableinfo` like this:

```
shell> mysql_tableinfo [options] db_name [db_like [tbl_like]]
```

The `db_name` argument indicates which database `mysql_tableinfo` should use as the location for the metadata tables. The database will be created if it does not exist. The tables will be named `db`, `tbl` (or `tbl_status`), `col`, and `idx`.

If the `db_like` or `tbl_like` arguments are given, they are used as patterns and metadata is generated only for databases or tables that match the patterns. These arguments default to `%` if not given.

Examples:

```
mysql_tableinfo info
mysql_tableinfo info world
mysql_tableinfo info mydb tmp%
```

Each of the commands stores information into tables in the `info` database. The first stores information for all databases and tables. The second stores information for all tables in the `world` database. The third stores information for tables in the `mydb` database that have names matching the pattern `tmp%`.

`mysql_tableinfo` supports the following options:

- `--help`

Display a help message and exit.

- `--clear`

Before populating each metadata table, drop it if it exists.

- `--clear-only`

Similar to `--clear`, but exits after dropping the metadata tables to be populated.

- `--col`

Generate column metadata into the `col` table.

- `--host=host_name, -h host_name`

Connect to the MySQL server on the given host.

- `--idx`

Generate index metadata into the `idx` table.

- `--password=password, -ppassword`

The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs. You can use an option file to avoid giving the password on the command line.

Specifying a password on the command line should be considered insecure. See [Section 5.8.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--prefix=prefix_str`

Add `prefix_str` at the beginning of each metadata table name.

- `--quiet, -q`

Be silent except for errors.

- `--socket=path, -S path`

The Unix socket file to use for the connection.

- `--tbl-status`

Use `SHOW TABLE STATUS` instead of `SHOW TABLES`. This provides more complete information, but is slower.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

7.23. `mysql_waitpid` — Kill Process and Wait for Its Termination

`mysql_waitpid` signals a process to terminate and waits for the process to exit. It uses the `kill()` system call and Unix signals, so it runs on Unix and Unix-like systems.

Invoke `mysql_waitpid` like this:

```
shell> mysql_waitpid [options] pid wait_time
```

`mysql_waitpid` sends signal 0 to the process identified by `pid` and waits up to `wait_time` seconds for the process to terminate. `pid` and `wait_time` must be positive integers.

If process termination occurs within the wait time or the process does not exist, `mysql_waitpid` returns 0. Otherwise, it returns 1.

If the `kill()` system call cannot handle signal 0, `mysql_waitpid()` uses signal 1 instead.

`mysql_waitpid` understands the following options:

- `--help, -?, -I`

Display a help message and exit.

- `--verbose, -v`

Verbose mode. Display a warning if signal 0 could not be used and signal 1 is used instead.

- `--version, -V`

Display version information and exit.

7.24. `mysql_zap` — Kill Processes That Match a Pattern

`mysql_zap` kills processes that match a pattern. It uses the `ps` command and Unix signals, so it runs on Unix and Unix-like systems.

Invoke `mysql_zap` like this:

```
shell> mysql_zap [-signal] [-?Ift] pattern
```

A process matches if its output line from the `ps` command contains the pattern. By default, `mysql_zap` asks for confirmation for each process. Respond `y` to kill the process, or `q` to exit `mysql_zap`. For any other response, `mysql_zap` does not attempt to kill the process.

If the `-signal` option is given, it specifies the name or number of the signal to send to each process. Otherwise, `mysql_zap` tries first with `TERM` (signal 15) and then with `KILL` (signal 9).

`mysql_zap` understands the following additional options:

- `--help, -?, -I`
Display a help message and exit.
- `-f`
Force mode. `mysql_zap` attempts to kill each process without confirmation.
- `-t`
Test mode. Display information about each process but do not kill it.

7.25. `perror` — Explain Error Codes

For most system errors, MySQL displays, in addition to an internal text message, the system error code in one of the following styles:

```
message ... (errno: #)
message ... (Errcode: #)
```

You can find out what the error code means by examining the documentation for your system or by using the `perror` utility.

`perror` prints a description for a system error code or for a storage engine (table handler) error code.

Invoke `perror` like this:

```
shell> perror [options] errorcode ...
```

Example:

```
shell> perror 13 64
OS error code 13: Permission denied
OS error code 64: Machine is not on the network
```

To obtain the error message for a MySQL Cluster error code, invoke `perror` with the `--ndb` option:

```
shell> perror --ndb errorcode
```

Note that the meaning of system error messages may be dependent on your operating system. A given error code may mean different things on different operating systems.

`perror` supports the following options:

- `--help, --info, -I, -?`

Display a help message and exit.

- `--ndb`

Print the error message for a MySQL Cluster error code.

- `--silent, -s`

Silent mode. Print only the error message.

- `--verbose, -v`

Verbose mode. Print error code and message. This is the default behavior.

- `--version, -V`

Display version information and exit.

7.26. `replace` — A String-Replacement Utility

The `replace` utility program changes strings in place in files or on the standard input.

Invoke `replace` in one of the following ways:

```
shell> replace from to [from to] ... -- file [file] ...
shell> replace from to [from to] ... < file
```

`from` represents a string to look for and `to` represents its replacement. There can be one or more pairs of strings.

Use the `--` option to indicate where the string-replacement list ends and the filenames begin. In this case, any file named on the command line is modified in place, so you may want to make a copy of the original before converting it. `replace` prints a message indicating which of the input files it actually modifies.

If the `--` option is not given, `replace` reads the standard input and writes to the standard output.

`replace` uses a finite state machine to match longer strings first. It can be used to swap strings. For example, the following command swaps `a` and `b` in the given files, `file1` and `file2`:

```
shell> replace a b b a -- file1 file2 ...
```

The `replace` program is used by `mysql2mysql`. See [Section 25.9.1, “mysql2mysql — Convert mSQL Programs for Use with MySQL”](#).

`replace` supports the following options:

- `-, -I`

Display a help message and exit.

- `-# debug_options`

Write a debugging log. The `debug_options` string often is `'d:t:o,file_name'`.

- `-s`

Silent mode. Print less information what the program does.

- `-v`

Verbose mode. Print more information about what the program does.

- `-V`

Display version information and exit.

7.27. **resolveip** — Resolve Hostname to IP Address or Vice Versa

The **resolveip** utility resolves hostnames to IP addresses and vice versa.

Invoke **resolveip** like this:

```
shell> resolveip [options] {host_name|ip-addr} ...
```

resolveip understands the options described in the following list.

- **--help, -info, -?, -I**

Display a help message and exit.

- **--silent, -s**

Silent mode. Produce less output.

- **--version, -V**

Display version information and exit.

7.28. **resolve_stack_dump** — Resolve Numeric Stack Trace Dump to Symbols

resolve_stack_dump resolves a numeric stack dump to symbols.

Invoke **resolve_stack_dump** like this:

```
shell> resolve_stack_dump [options] symbols_file [numeric_dump_file]
```

The symbols file should include the output from the **nm --numeric-sort mysqld** command. The numeric dump file should contain a numeric stack track from **mysqld**. If no numeric dump file is named on the command line, the stack trace is read from the standard input.

resolve_stack_dump understands the options described in the following list.

- **--help, -h**

Display a help message and exit.

- **--numeric-dump-file=file_name, -n file_name**

Read the stack trace from the given file.

- **--symbols-file=file_name, -s file_name**

Use the given symbols file.

- **--version, -V**

Display version information and exit.

Chapter 8. Language Structure

This chapter discusses the rules for writing the following elements of SQL statements when using MySQL:

- Literal values such as strings and numbers
- Identifiers such as database, table, and column names
- Reserved words
- User-defined and system variables
- Comments

8.1. Literal Values

This section describes how to write literal values in MySQL. These include strings, numbers, hexadecimal values, boolean values, and [NULL](#). The section also covers the various nuances and “gotchas” that you may run into when dealing with these basic types in MySQL.

8.1.1. Strings

A string is a sequence of bytes or characters, enclosed within either single quote (“’”) or double quote (“””) characters. Examples:

```
'a string'
"another string"
```

If the [ANSI_QUOTES](#) SQL mode is enabled, string literals can be quoted only within single quotes because a string quoted within double quotes is interpreted as an identifier.

A *binary string* is a string of bytes that has no character set or collation. A *non-binary string* is a string of characters that has a character set and collation. For both types of strings, comparisons are based on the numeric values of the string unit. For binary strings, the unit is the byte. For non-binary strings the unit is the character and some character sets allow multi-byte characters. Character value ordering is a function of the string collation.

String literals may have an optional character set introducer and [COLLATE](#) clause:

```
[_charset_name] 'string' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

You can use [N'*literal*'](#) (or [n'*literal*'](#)) to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For more information about these forms of string syntax, see [Section 9.3.5, “Character String Literal Character Set and Collation”](#), and [Section 9.3.6, “National Character Set”](#).

Within a string, certain sequences have special meaning. Each of these sequences begins with a backslash (“\”), known as the *escape character*. MySQL recognizes the following escape sequences:

\0	An ASCII 0 (NUL) character.
\'	A single quote (“’”) character.
\"	A double quote (“””) character.
\b	A backspace character.

<code>\n</code>	A newline (linefeed) character.
<code>\r</code>	A carriage return character.
<code>\t</code>	A tab character.
<code>\Z</code>	ASCII 26 (Control-Z). See note following the table.
<code>\\</code>	A backslash (“\”) character.
<code>\%</code>	A “%” character. See note following the table.
<code>_</code>	A “_” character. See note following the table.

For all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped. For example, “`\x`” is just “`x`”.

These sequences are case sensitive. For example, “`\b`” is interpreted as a backspace, but “`\B`” is interpreted as “`B`”.

The ASCII 26 character can be encoded as “`\Z`” to enable you to work around the problem that ASCII 26 stands for END-OF-FILE on Windows. ASCII 26 within a file causes problems if you try to use `mysql db_name < file_name`.

Escape processing is done according to the character set indicated by the `character_set_connection` system variable. This is true even for strings that are preceded by an introducer that indicates a different character set, as discussed in [Section 9.3.5, “Character String Literal Character Set and Collation”](#).

The “`\%`” and “`\_`” sequences are used to search for literal instances of “%” and “_” in pattern-matching contexts where they would otherwise be interpreted as wildcard characters. See the description of the [LIKE](#) operator in [Section 11.4.1, “String Comparison Functions”](#). If you use “`\%`” or “`\_`” in non-pattern-matching contexts, they evaluate to the strings “`\%`” and “`\_`”, not to “%” and “_”.

There are several ways to include quote characters within a string:

- A “`'`” inside a string quoted with “`'`” may be written as “`''`”.
- A “`”` inside a string quoted with “`”` may be written as “`""`”.
- Precede the quote character by an escape character (“`\`”).
- A “`'`” inside a string quoted with “`”`” needs no special treatment and need not be doubled or escaped. In the same way, “`”`” inside a string quoted with “`'`” needs no special treatment.

The following [SELECT](#) statements demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel'lo', '\hello';
+-----+-----+-----+-----+
| hello | "hello" | ""hello"" | hel'lo | 'hello' |
+-----+-----+-----+-----+

mysql> SELECT "hello", '"hello"', '""hello""', "hel"lo", "\"hello";
+-----+-----+-----+-----+
| hello | 'hello' | '""hello"" | hel"lo | "\"hello' |
+-----+-----+-----+-----+

mysql> SELECT 'This\nIs\nFour\nLines';
+-----+
| This
Is
Four
Lines |
+-----+

mysql> SELECT 'disappearing\ backslash';
+-----+
| disappearing backslash |
+-----+
```

If you want to insert binary data into a string column (such as a [BLOB](#) column), the following characters must be represented by escape sequences:

NUL	NUL byte (ASCII 0). Represent this character by “ <code>\0</code> ” (a backslash followed by an ASCII “0” character).

<code>\</code>	Backslash (ASCII 92). Represent this character by “ <code>\\</code> ”.
<code>'</code>	Single quote (ASCII 39). Represent this character by “ <code>\'</code> ”.
<code>"</code>	Double quote (ASCII 34). Represent this character by “ <code>\"</code> ”.

When writing application programs, any string that might contain any of these special characters must be properly escaped before the string is used as a data value in an SQL statement that is sent to the MySQL server. You can do this in two ways:

- Process the string with a function that escapes the special characters. In a C program, you can use the `mysql_real_escape_string()` C API function to escape characters. See [Section 25.2.3.53](#), “`mysql_real_escape_string()`”. The Perl DBI interface provides a `quote` method to convert special characters to the proper escape sequences. See [Section 25.4](#), “MySQL Perl API”. Other language interfaces may provide a similar capability.
- As an alternative to explicitly escaping special characters, many MySQL APIs provide a placeholder capability that enables you to insert special markers into a statement string, and then bind data values to them when you issue the statement. In this case, the API takes care of escaping special characters in the values for you.

8.1.2. Numbers

Integers are represented as a sequence of digits. Floats use “.” as a decimal separator. Either type of number may be preceded by “-” or “+” to indicate a negative or positive value, respectively

Examples of valid integers:

```
1221
0
-32
```

Examples of valid floating-point numbers:

```
294.42
-32032.6809e+10
148.00
```

An integer may be used in a floating-point context; it is interpreted as the equivalent floating-point number.

8.1.3. Hexadecimal Values

MySQL supports hexadecimal values. In numeric contexts, these act like integers (64-bit precision). In string contexts, these act like binary strings, where each pair of hex digits is converted to a character:

```
mysql> SELECT x'4D7953514C';
-> 'MySQL'
mysql> SELECT 0x0a+0;
-> 10
mysql> SELECT 0x5061756c;
-> 'Paul'
```

The default type of a hexadecimal value is a string. If you want to ensure that the value is treated as a number, you can use `CAST( . . . AS UNSIGNED)`:

```
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
-> 'A', 65
```

The `x'hexstring'` syntax is based on standard SQL. The `0x` syntax is based on ODBC. Hexadecimal strings are often used by ODBC to supply values for `BLOB` columns.

You can convert a string or a number to a string in hexadecimal format with the `HEX()` function:

```
mysql> SELECT HEX('cat');
-> '636174'
mysql> SELECT 0x636174;
-> 'cat'
```

8.1.4. Boolean Values

The constants `TRUE` and `FALSE` evaluate to `1` and `0`, respectively. The constant names can be written in any lettercase.

```
mysql> SELECT TRUE, true, FALSE, false;
-> 1, 1, 0, 0
```

8.1.5. Bit-Field Values

Bit-field values can be written using `b'value'` or `0bvalue` notation. `value` is a binary value written using zeros and ones.

Bit-field notation is convenient for specifying values to be assigned to `BIT` columns:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
mysql> INSERT INTO t SET b = b'0101';
```

Bit values are returned as binary values. To display them in printable form, add `0` or use a conversion function such as `BIN()`. Note that high-order `0` bits are not displayed in the converted value.

```
mysql> SELECT b+0, BIN(b+0), OCT(b+0), HEX(b+0) FROM t;
```

b+0	BIN(b+0)	OCT(b+0)	HEX(b+0)
255	11111111	377	FF
10	1010	12	A
5	101	5	5

Bit values assigned to user variables are treated as binary strings. To assign a bit value as a number to a user variable, use `CAST()`:

```
mysql> SET @v1 = b'1000001', @v2 = CAST(b'1000001' AS UNSIGNED);
mysql> SELECT @v1, @v2;
```

@v1	@v2
A	65

8.1.6. NULL Values

The `NULL` value means “no data.” `NULL` can be written in any lettercase. A synonym is `\N` (case sensitive).

Be aware that the `NULL` value is different from values such as `0` for numeric types or the empty string for string types. See [Section B.1.5.3, “Problems with NULL Values”](#).

For text file import or export operations performed with `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, `NULL` is represented by the `\N` sequence. See [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).

8.2. Identifiers

Database, table, index, column, and alias names are identifiers. This section describes the allowable syntax for identifiers in MySQL.

The following table describes the maximum length for each type of identifier.

Identifier	Maximum Length (bytes)
Database	64
Table	64
Column	64
Index	64
Alias	255

There are some restrictions on the characters that may appear in identifiers:

- No identifier can contain ASCII 0 (0x00) or a byte with a value of 255.
- The use of identifier quote characters in identifiers is permitted, although it is best to avoid doing so if possible.
- Database, table, and column names should not end with space characters.
- Before MySQL 5.1.6, database names cannot contain “/”, “\”, “.”, or characters that are not allowed in a directory name.
- Before MySQL 5.1.6, table names cannot contain “/”, “\”, “.”, or characters that are not allowed in a filename.
- The length of the identifier is in bytes, not characters. If you use multi-byte characters in your identifier names, then the maximum length will depend on the byte count of all the characters used.

As of MySQL 5.1.6, special characters in database and table names are encoded in the corresponding filesystem names as described in [Section 8.2.3, “Mapping of Identifiers to Filenames”](#). If you have databases or tables from an older version of MySQL that contain special characters and that have not been updated to use the new encoding, you will see their names displayed with a prefix of `#mysql150#`. For information about referring to such names or converting them to the newer encoding, see that section.

Identifiers are stored using Unicode (UTF-8). This applies to identifiers in table definitions that are stored in `.frm` files and to identifiers stored in the grant tables in the `mysql` database. The sizes of the string columns in the grant tables (and in any other tables) in MySQL 5.1 are given as number of characters. This means that (unlike some earlier versions of MySQL) you can use multi-byte characters without reducing the number of characters allowed for values stored in these columns.

An identifier may be quoted or unquoted. If an identifier is a reserved word or contains special characters, you *must* quote it whenever you refer to it. (Exception: A word that follows a period in a qualified name must be an identifier, so it need not be quoted even if it is reserved.) For a list of reserved words, see [Section 8.3, “Reserved Words”](#). Special characters are those outside the set of alphanumeric characters from the current character set, “_”, and “\$”.

The identifier quote character is the backtick (“`”):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

If the `ANSI_QUOTES` SQL mode is enabled, it is also allowable to quote identifiers within double quotes:

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax. (...)
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

Note: Because the `ANSI_QUOTES` mode causes the server to interpret double-quoted strings as identifiers, string literals must be enclosed within single quotes when this mode is enabled. They cannot be enclosed within double quotes.

The server SQL mode is controlled as described in [Section 5.2.6, “SQL Modes”](#).

Identifier quote characters can be included within an identifier *if you quote the identifier*. If the character to be included within the identifier is the same as that used to quote the identifier itself, then you need to double the character. The following statement creates a table named `a`b` that contains a column named `c`d`:

```
mysql> CREATE TABLE `a``b` (`c`d` INT);
```

Identifiers may begin with a digit but unless quoted may not consist solely of digits.

It is recommended that you do not use names of the form `Me` or `MeN`, where `M` and `N` are integers. For example, avoid using `1e` or `2e2` as identifiers, because an expression such as `1e+3` is ambiguous. Depending on context, it might be interpreted as the expression `1e + 3` or as the number `1e+3`.

A user variable cannot be used directly in an SQL statement as an identifier or as part of an identifier. See [Section 8.4, “User-Defined Variables”](#), for more information and examples of workarounds.

Be careful when using `MD5()` to produce table names because it can produce names in illegal or ambiguous formats such as those just described.

8.2.1. Identifier Qualifiers

MySQL allows names that consist of a single identifier or multiple identifiers. The components of a multiple-part name should be separated by period (“.”) characters. The initial parts of a multiple-part name act as qualifiers that affect the context within which the final identifier is interpreted.

In MySQL you can refer to a column using any of the following forms:

Column Reference	Meaning
<i>col_name</i>	The column <i>col_name</i> from whichever table used in the statement contains a column of that name.
<i>tbl_name.col_name</i>	The column <i>col_name</i> from table <i>tbl_name</i> of the default database.
<i>db_name.tbl_name.col_name</i>	The column <i>col_name</i> from table <i>tbl_name</i> of the database <i>db_name</i> .

If any components of a multiple-part name require quoting, quote them individually rather than quoting the name as a whole. For example, write ``my-table`.`my-column``, not ``my-table.my-column``.

You need not specify a *tbl_name* or *db_name.tbl_name* prefix for a column reference in a statement unless the reference would be ambiguous. Suppose that tables *t1* and *t2* each contain a column *c*, and you retrieve *c* in a `SELECT` statement that uses both *t1* and *t2*. In this case, *c* is ambiguous because it is not unique among the tables used in the statement. You must qualify it with a table name as *t1.c* or *t2.c* to indicate which table you mean. Similarly, to retrieve from a table *t* in database *db1* and from a table *t* in database *db2* in the same statement, you must refer to columns in those tables as *db1.t.col_name* and *db2.t.col_name*.

A word that follows a period in a qualified name must be an identifier, so it is not necessary to quote it, even if it is a reserved word.

The syntax `.tbl_name` means the table *tbl_name* in the default database. This syntax is accepted for ODBC compatibility because some ODBC programs prefix table names with a “.” character.

8.2.2. Identifier Case Sensitivity

In MySQL, databases correspond to directories within the data directory. Each table within a database corresponds to at least one file within the database directory (and possibly more, depending on the storage engine). Consequently, the case sensitivity of the underlying operating system determines the case sensitivity of database and table names. This means database and table names are case sensitive in most varieties of Unix, and not case sensitive in Windows. One notable exception is Mac OS X, which is Unix-based but uses a default filesystem type (HFS+) that is not case sensitive. However, Mac OS X also supports UFS volumes, which are case sensitive just as on any Unix. See [Section 1.9.4, “MySQL Extensions to Standard SQL”](#). The `lower_case_table_names` system variable also affects how the server handles identifier case sensitivity, as described later in this section.

MySQL Enterprise

`lower_case_table_names` is just one of the system variables monitored by the MySQL Enterprise Monitor. For information about subscribing to this service see, <http://www.mysql.com/products/enterprise/advisors.html>.

Note

Although database and table names are not case sensitive on some platforms, you should not refer to a given database or table using different cases within the same statement. The following statement would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Column, index and stored routine names are not case sensitive on any platform, nor are column aliases. Trigger names are case sensitive.

By default, table aliases are case sensitive on Unix, but not so on Windows or Mac OS X. The following statement would not work on Unix, because it refers to the alias both as `a` and as `A`:

```
mysql> SELECT col_name FROM tbl_name AS a
-> WHERE a.col_name = 1 OR A.col_name = 2;
```

However, this same statement is permitted on Windows. To avoid problems caused by such differences, it is best to adopt a consistent convention, such as always creating and referring to databases and tables using lowercase names. This convention is recommended for maximum portability and ease of use.

How table and database names are stored on disk and used in MySQL is affected by the `lower_case_table_names` system variable, which you can set when starting `mysqld`. `lower_case_table_names` can take the values shown in the following table. On Unix, the default value of `lower_case_table_names` is 0. On Windows the default value is 1. On Mac OS X, the default value is 2.

Value	Meaning
0	Table and database names are stored on disk using the lettercase specified in the <code>CREATE TABLE</code> or <code>CREATE DATABASE</code> statement. Name comparisons are case sensitive. Note that if you force this variable to 0 with <code>-lower-case-table-names=0</code> on a case-insensitive filesystem and access <code>MyISAM</code> table names using different lettercases, index corruption may result.
1	Table names are stored in lowercase on disk and name comparisons are not case sensitive. MySQL converts all table names to lowercase on storage and lookup. This behavior also applies to database names and table aliases.
2	Table and database names are stored on disk using the lettercase specified in the <code>CREATE TABLE</code> or <code>CREATE DATABASE</code> statement, but MySQL converts them to lowercase on lookup. Name comparisons are not case sensitive. Note: This works <i>only</i> on filesystems that are not case sensitive! <code>InnoDB</code> table names are stored in lowercase, as for <code>lower_case_table_names=1</code> .

If you are using MySQL on only one platform, you don't normally have to change the `lower_case_table_names` variable. However, you may encounter difficulties if you want to transfer tables between platforms that differ in filesystem case sensitivity. For example, on Unix, you can have two different tables named `my_table` and `MY_TABLE`, but on Windows these two names are considered identical. To avoid data transfer problems stemming from lettercase of database or table names, you have two options:

- Use `lower_case_table_names=1` on all systems. The main disadvantage with this is that when you use `SHOW TABLES` or `SHOW DATABASES`, you don't see the names in their original lettercase.
- Use `lower_case_table_names=0` on Unix and `lower_case_table_names=2` on Windows. This preserves the lettercase of database and table names. The disadvantage of this is that you must ensure that your statements always refer to your database and table names with the correct lettercase on Windows. If you transfer your statements to Unix, where lettercase is significant, they do not work if the lettercase is incorrect.

Exception: If you are using `InnoDB` tables and you are trying to avoid these data transfer problems, you should set `lower_case_table_names` to 1 on all platforms to force names to be converted to lowercase.

Note that if you plan to set the `lower_case_table_names` system variable to 1 on Unix, you must first convert your old database and table names to lowercase before restarting `mysqld` with the new variable setting.

Object names may be considered duplicates if their uppercase forms are equal according to a binary collation. That is true for names of cursors, conditions, functions, procedures, savepoints, and routine local variables. It is not true for names of columns, constraints, databases, partitions, statements prepared with `PREPARE`, tables, triggers, users, and user-defined variables.

8.2.3. Mapping of Identifiers to Filenames

There is a correspondence between database and table identifiers and names in the filesystem. MySQL represents each database as a directory in the data directory, and each table by one or more files in the appropriate database directory.

Before MySQL 5.1.6, there are some limitations on the characters that can be used in identifiers for database objects that correspond to filesystem objects. For example, pathname separator characters are disallowed, and `“.”` is disallowed because it begins the extension for table files.

As of MySQL 5.1.6, any character is legal in database or table identifiers except ASCII NUL (`0x00`). MySQL encodes any characters that are problematic in the corresponding filesystem objects when it creates database directories or table files:

- Basic Latin letters (`a..zA..Z`) and digits (`0..9`) are encoded as is. Consequently, their case sensitivity directly depends on filesystem features.
- All other national letters from alphabets that have uppercase/lowercase mapping are encoded as follows:

Code range	Pattern	Number	Used	Unused	Blocks
00C0..017F	[@][0..4][g..z]	5*20= 100	97	3	Latin1 Supplement + Ext A

0370..03FF	[@][5..9][g..z]	5*20=	100	88	12	Greek + Coptic
0400..052F	[@][g..z][0..6]	20*7=	140	140	137	Cyrillic
0530..058F	[@][g..z][7..8]	20*2=	40	38	2	Armenian
2160..217F	[@][g..z][9]	20*1=	20	16	4	Number Forms
0180..02AF	[@][g..z][a..k]	28*11=	220	203	17	Latin Ext B + IPA
1E00..0EFF	[@][g..z][l..r]	20*7=	140	136	4	Latin Additional Extended
1F00..1FFF	[@][g..z][s..z]	20*8=	160	144	16	Greek Extended
....	[@][a..f][g..z]	6*20=	120	0	120	RESERVED
24B6..24E9	[@][@][a..z]		26	26	0	Enclosed Alphanumerics
FF21..FF5A	[@][a..z][@]		26	26	0	Full Width forms

One of the bytes in the sequence encodes lettercase. For example: **LATIN CAPITAL LETTER A WITH GRAVE** is encoded as **@0G**, whereas **LATIN SMALL LETTER A WITH GRAVE** is encoded as **@0g**. Here the third byte (**G** or **g**) indicates lettercase. (On a case-insensitive filesystem, both letters will be treated as the same.)

For some blocks, such as Cyrillic, the second byte determines lettercase. For other blocks, such as Latin1 Supplement, the third byte determines lettercase. If two bytes in the sequence are letters (as in Greek Extended), the leftmost letter character stands for lettercase. All other letter bytes must be in lowercase.

- All non-letter characters, as well as letters from alphabets that do not have uppercase/lowercase mapping (such as Hebrew) are encoded using hexadecimal representation using lowercase letters for hex digits **a..f**:

```
0x003F -> @003f
0xFFFF -> @ffff
```

The hexadecimal values correspond to character values in the **ucs2** double-byte character set.

On Windows, some names such as **nul**, **prn**, and **aux** cannot be used as filenames because they are reserved as device names. As of MySQL 5.1.10, these are allowable names in MySQL. They are encoded by appending **@@@** to the name when the server creates the corresponding file or directory. This occurs on all platforms for portability of the corresponding database object between platforms.

If you have databases or tables from a version of MySQL older than 5.1.6 that contain special characters and that have not been updated to use the new encoding, you will see their names displayed with a prefix of **#mysql50#** in the output from **INFORMATION_SCHEMA** tables or **SHOW** statements. For example, if you have a table named **a@b** and its name encoding has not been updated, **SHOW TABLES** displays it like this:

```
mysql> SHOW TABLES;
+-----+
| Tables_in_test |
+-----+
| #mysql50#a@b  |
+-----+
```

To refer to such a name for which the encoding has not been updated, you must supply the **#mysql50#** prefix:

```
mysql> SHOW COLUMNS FROM `a@b`;
ERROR 1146 (42S02): Table 'test.a@b' doesn't exist

mysql> SHOW COLUMNS FROM `#mysql50#a@b`;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| i     | int(11) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
```

To update old names to eliminate the need to use the special prefix to refer to them, re-encode them with **mysqlcheck**. The following command updates all names to the new encoding:

```
shell> mysqlcheck --check-upgrade --fix-db-names --fix-table-names --all-databases
```

To check only specific databases or tables, omit **--all-databases** and provide the appropriate database or table arguments. For information about **mysqlcheck** invocation syntax, see [Section 7.12, “mysqlcheck — A Table Maintenance and Repair Program”](#).

8.2.4. Function Name Parsing and Resolution

MySQL 5.1 supports built-in (native) functions, user-defined functions (UDFs), and stored functions. This section describes how the server recognizes whether the name of a built-in function is used as a function call or as an identifier, and how the server determines

which function to use in cases when functions of different types exist with a given name.

Built-In Function Name Parsing

The parser uses default rules for parsing names of built-in functions. These rules can be changed by enabling the `IGNORE_SPACE` SQL mode.

When the parser encounters a word that is the name of a built-in function, it must determine whether the name signifies a function call or is instead a non-expression reference to an identifier such as a table or column name. For example, in the following statements, the first reference to `COUNT` is a function call, whereas the second reference is a table name:

```
SELECT COUNT(*) FROM mytable;
CREATE TABLE count (i INT);
```

The parser should recognize the name of a built-in function as indicating a function call only when parsing what is expected to be an expression. That is, in non-expression context, function names are permitted as identifiers.

However, some built-in functions have special parsing or implementation considerations, so the parser uses the following rules by default to distinguish whether their names are being used as function calls or as identifiers in non-expression context:

- To use the name as a function call in an expression, there must be no whitespace between the name and the following “(” parenthesis character.
- Conversely, to use the function name as an identifier, it must not be followed immediately by a parenthesis.

The requirement that function calls be written with no whitespace between the name and the parenthesis applies only to the built-in functions that have special considerations. `COUNT` is one such name. The exact list of function names for which following whitespace determines their interpretation are those listed in the `sql_functions[]` array of the `sql/lex.h` source file. Before MySQL 5.1, they are rather numerous (about 200), so you may find it easiest to treat the no-whitespace requirement as applying to all function calls. In MySQL 5.1, parser improvements reduce to about 30 the number of affected function names.

For functions not listed in the `sql_functions[]` array, whitespace does not matter. They are interpreted as function calls only when used in expression context and may be used freely as identifiers otherwise. `ASCII` is one such name. However, for these non-affected function names, interpretation may vary in expression context: `func_name ()` is interpreted as a built-in function if there is one; if not, `func_name ()` is interpreted as a user-defined function or stored function if one exists with that name.

The `IGNORE_SPACE` SQL mode can be used to modify how the parser treats function names that are whitespace-sensitive:

- With `IGNORE_SPACE` disabled, the parser interprets the name as a function call when there is no whitespace between the name and the following parenthesis. This occurs even when the function name is used in non-expression context:

```
mysql> CREATE TABLE count(i INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'count(i INT)'
```

To eliminate the error and cause the name to be treated as an identifier, either use whitespace following the name or write it as a quoted identifier (or both):

```
CREATE TABLE count (i INT);
CREATE TABLE `count` (i INT);
CREATE TABLE `count` (i INT);
```

- With `IGNORE_SPACE` enabled, the parser loosens the requirement that there be no whitespace between the function name and the following parenthesis. This provides more flexibility in writing function calls. For example, either of the following function calls are legal:

```
SELECT COUNT(*) FROM mytable;
SELECT COUNT (*) FROM mytable;
```

However, enabling `IGNORE_SPACE` also has the side effect that the parser treats the affected function names as reserved words (see [Section 8.3, “Reserved Words”](#)). This means that a space following the name no longer signifies its use as an identifier. The name can be used in function calls with or without following whitespace, but causes a syntax error in non-expression context unless it is quoted. For example, with `IGNORE_SPACE` enabled, both of the following statements fail with a syntax error because the pars-

er interprets `count` as a reserved word:

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

To use the function name in non-expression context, write it as a quoted identifier:

```
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

To enable the `IGNORE_SPACE` SQL mode, use this statement:

```
SET sql_mode = 'IGNORE_SPACE';
```

`IGNORE_SPACE` is also enabled by certain other composite modes such as `ANSI` that include it in their value:

```
SET sql_mode = 'ANSI';
```

Check [Section 5.2.6, “SQL Modes”](#), to see which composite modes enable `IGNORE_SPACE`.

To minimize the dependency of SQL code on the `IGNORE_SPACE` setting, use these guidelines:

- Avoid creating UDFs or stored functions that have the same name as a built-in function.
- Avoid using function names in non-expression context. For example, these statements use `count` (one of the affected function names affected by `IGNORE_SPACE`), so they fail with or without whitespace following the name if `IGNORE_SPACE` is enabled:

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

If you must use a function name in non-expression context, write it as a quoted identifier:

```
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

The number of function names affected by `IGNORE_SPACE` was reduced significantly in MySQL 5.1.13, from about 200 to about 30. As of MySQL 5.1.13, only the following functions are still affected by the `IGNORE_SPACE` setting:

ADDDATE	BIT_AND	BIT_OR
BIT_XOR	CAST	COUNT
CURDATE	CURTIME	DATE_ADD
DATE_SUB	EXTRACT	GROUP_CONCAT
MAX	MID	MIN
NOW	POSITION	SESSION_USER
STD	STDDEV	STDDEV_POP
STDDEV_SAMP	SUBDATE	SUBSTR
SUBSTRING	SUM	SYSDATE
SYSTEM_USER	TRIM	VARIANCE
VAR_POP	VAR_SAMP	

For earlier versions of MySQL, check the contents of the `sql_functions[]` array in the `sql/lex.h` source file to see which functions are affected by `IGNORE_SPACE`.

Incompatibility warning: The change in MySQL 5.1.13 that reduces the number of function names affected by `IGNORE_SPACE` improves the consistency of parser operation. However, it also introduces the possibility of incompatibility for old SQL code that relies on the following conditions:

- `IGNORE_SPACE` is disabled.
- The presence or absence of whitespace following a function name is used to distinguish between a built-in function and stored function that have the same name (for example, `PI()` versus `PI ()`).

For functions that are no longer affected by `IGNORE_SPACE` as of MySQL 5.1.13, that strategy no longer works. Either of the following approaches can be used if you have code that is subject to the preceding incompatibility:

- If a stored function has a name that conflicts with a built-in function, refer to the stored function with a schema name qualifier, regardless of whether whitespace is present. For example, write `schema_name.PI()` or `schema_name.PI ()`.
- Alternatively, rename the stored function to use a non-conflicting name and change invocations of the function to use the new name.

Function Name Resolution

The following rules describe how the server resolves references to function names for function creation and invocation:

- Built-in functions and user-defined functions

As of MySQL 5.1.14, an error occurs if you try to create a UDF with the same name as a built-in function. Before 5.1.14, a UDF can be created with the same name as a built-in function but the UDF cannot be invoked because the parser resolves invocations of the function to refer to the built-in function. For example, if you create a UDF named `ABS`, references to `ABS()` invoke the built-in function.

- Built-in functions and stored functions

It is possible to create a stored function with the same name as a built-in function, but to invoke the stored function it is necessary to qualify it with a schema name. For example, if you create a stored function named `PI` in the `test` schema, you invoke it as `test.PI()` because the server resolves `PI()` as a reference to the built-in function. As of 5.1.14, the server creates a warning if the stored function name collides with a built-in function name. The warning can be displayed with `SHOW WARNINGS`.

- User-defined functions and stored functions

User-defined functions and stored functions share the same namespace, so you cannot create a UDF and a stored function with the same name.

The preceding function name resolution rules have implications for upgrading to versions of MySQL that implement new built-in functions:

- If you have already created a user-defined function with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF with a different non-conflicting name.
- If a new version of MySQL implements a built-in function with the same name as an existing stored function, you have two choices: Rename the stored function to use a non-conflicting name, or change calls to the function so that they use a schema qualifier (that is, use `schema_name.func_name()` syntax).

8.3. Reserved Words

Certain words such as `SELECT`, `DELETE`, or `BIGINT` are reserved and require special treatment for use as identifiers such as table and column names. This may also be true for the names of built-in functions.

Reserved words are permitted as identifiers if you quote them as described in [Section 8.2, “Identifiers”](#):

```
mysql> CREATE TABLE interval (begin INT, end INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'interval (begin INT, end INT)'

mysql> CREATE TABLE `interval` (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Exception: A word that follows a period in a qualified name must be an identifier, so it need not be quoted even if it is reserved:

```
mysql> CREATE TABLE mydb.interval (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Names of built-in functions are permitted as identifiers but may require care to be used as such. For example, `COUNT` is acceptable as a column name. However, by default, no whitespace is allowed in function invocations between the function name and the following “(” character. This requirement enables the parser to distinguish whether the name is used in a function call or in non-function context. For further detail on recognition of function names, see [Section 8.2.4, “Function Name Parsing and Resolution”](#).

The words in the following table are explicitly reserved in MySQL 5.1. At some point, you might upgrade to a higher version, so it's a good idea to have a look at future reserved words, too. You can find these in the manuals that cover higher versions of MySQL. Most of the words in the table are forbidden by standard SQL as column or table names (for example, `GROUP`). A few are reserved because MySQL needs them and uses a `yacc` parser. A reserved word can be used as an identifier if you quote it.

ACCESSIBLE	ADD	ALL
ALTER	ANALYZE	AND
AS	ASC	ASENSITIVE
BEFORE	BETWEEN	BIGINT
BINARY	BLOB	BOTH
BY	CALL	CASCADE
CASE	CHANGE	CHAR
CHARACTER	CHECK	COLLATE
COLUMN	CONDITION	CONSTRAINT
CONTINUE	CONVERT	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DATABASE	DATABASES	DAY_HOUR
DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND
DEC	DECIMAL	DECLARE
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DIV
DOUBLE	DROP	DUAL
EACH	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FLOAT	FLOAT4
FLOAT8	FOR	FORCE
FOREIGN	FROM	FULLTEXT
GRANT	GROUP	HAVING
HIGH_PRIORITY	HOURL_MICROSECOND	HOURL_MINUTE
HOURL_SECOND	IF	IGNORE
IN	INDEX	INFILE
INNER	INOUT	INSENSITIVE
INSERT	INT	INT1
INT2	INT3	INT4
INT8	INTEGER	INTERVAL
INTO	IS	ITERATE
JOIN	KEY	KEYS

KILL	LEADING	LEAVE
LEFT	LIKE	LIMIT
LINEAR	LINES	LOAD
LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOOP	LOW_PRIORITY	MASTER_SSL_VERIFY_SERVER_CERT
MATCH	MEDIUMBLOB	MEDIUMINT
MEDIUMTEXT	MIDDLEINT	MINUTE_MICROSECOND
MINUTE_SECOND	MOD	MODIFIES
NATURAL	NOT	NO_WRITE_TO_BINLOG
NULL	NUMERIC	ON
OPTIMIZE	OPTION	OPTIONALLY
OR	ORDER	OUT
OUTER	OUTFILE	PRECISION
PRIMARY	PROCEDURE	PURGE
RANGE	READ	READS
READ_ONLY	READ_WRITE	REAL
REFERENCES	REGEXP	RELEASE
RENAME	REPEAT	REPLACE
REQUIRE	RESTRICT	RETURN
REVOKE	RIGHT	RLIKE
SCHEMA	SCHEMAS	SECOND_MICROSECOND
SELECT	SENSITIVE	SEPARATOR
SET	SHOW	SMALLINT
SPATIAL	SPECIFIC	SQL
SQLEXCEPTION	SQLSTATE	SQLWARNING
SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS	SQL_SMALL_RESULT
SSL	STARTING	STRAIGHT_JOIN
TABLE	TERMINATED	THEN
TINYBLOB	TINYINT	TINYTEXT
TO	TRAILING	TRIGGER
TRUE	UNDO	UNION
UNIQUE	UNLOCK	UNSIGNED
UPDATE	USAGE	USE
USING	UTC_DATE	UTC_TIME
UTC_TIMESTAMP	VALUES	VARBINARY
VARCHAR	VARCHARACTER	VARYING
WHEN	WHERE	WHILE
WITH	WRITE	X509
XOR	YEAR_MONTH	ZEROFILL

The following are new reserved words in MySQL 5.1:

ACCESSIBLE	LINEAR	MASTER_SSL_VERIFY_SERVER_CERT
RANGE	READ_ONLY	READ_WRITE

MySQL allows some keywords to be used as unquoted identifiers because many people previously used them. Examples are those in the following list:

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

8.4. User-Defined Variables

You can store a value in a user-defined variable and then refer to it later. This enables you to pass values from one statement to another. *User-defined variables are connection-specific.* That is, a user variable defined by one client cannot be seen or used by other clients. All variables for a given client connection are automatically freed when that client exits.

User variables are written as `@var_name`, where the variable name `var_name` may consist of alphanumeric characters from the current character set, “.”, “_”, and “\$”. The default character set is `latin1` (cp1252 West European). This may be changed with the `-character-set-server` option to `mysqld`. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#). A user variable name can contain other characters if you quote it as a string or identifier (for example, `@'my-var'`, `@"my-var"`, or `@`my-var``).

Note: User variable names not case sensitive in MySQL 5.0 and up.

One way to set a user-defined variable is by issuing a `SET` statement:

```
SET @var_name = expr [, @var_name = expr] ...
```

For `SET`, either `=` or `:=` can be used as the assignment operator. The `expr` assigned to each variable can evaluate to an integer, real, string, or `NULL` value. However, if the value of the variable is selected in a result set, it is returned to the client as a string.

You can also assign a value to a user variable in statements other than `SET`. In this case, the assignment operator must be `:=` and not `=` because `=` is treated as a comparison operator in non-`SET` statements:

```
mysql> SET @t1=0, @t2=0, @t3=0;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4, @t1, @t2, @t3;
+-----+-----+-----+
| @t1:=(@t2:=1)+@t3:=4 | @t1 | @t2 | @t3 |
+-----+-----+-----+
| 5 | 5 | 1 | 4 |
+-----+-----+-----+
```

User variables may be used in contexts where expressions are allowed. This does not currently include contexts that explicitly require a literal value, such as in the `LIMIT` clause of a `SELECT` statement, or the `IGNORE N LINES` clause of a `LOAD DATA` statement.

If a user variable is assigned a string value, it has the same character set and collation as the string. The coercibility of user variables is implicit. (This is the same coercibility as for table column values.)

If you refer to a variable that has not been initialized, it has a value of `NULL` and a type of string.

Bit values assigned to user variables are treated as binary strings. To assign a bit value as a number to a user variable, use `CAST()`:

```
mysql> SET @v1 = b'1000001', @v2 = CAST(b'1000001' AS UNSIGNED);
mysql> SELECT @v1, @v2;
+-----+-----+
| @v1 | @v2 |
+-----+-----+
| A | 65 |
+-----+-----+
```

Note

In a **SELECT** statement, each expression is evaluated only when sent to the client. This means that in a **HAVING**, **GROUP BY**, or **ORDER BY** clause, you cannot refer to an expression that involves variables that are set in the **SELECT** list. For example, the following statement does *not* work as expected:

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM tbl_name HAVING b=5;
```

The reference to **b** in the **HAVING** clause refers to an alias for an expression in the **SELECT** list that uses **@aa**. This does not work as expected: **@aa** contains the value of **id** from the previous selected row, not from the current row.

The order of evaluation for user variables is undefined and may change based on the elements contained within a given query. In **SELECT @a, @a := @a+1 ...**, you might think that MySQL will evaluate **@a** first and then do an assignment second, but changing the query (for example, by adding a **GROUP BY**, **HAVING**, or **ORDER BY** clause) may change the order of evaluation.

The general rule is never to assign a value to a user variable in one part of a statement *and* use the same variable in some other part of the same statement. You might get the results you expect, but this is not guaranteed.

Another issue with setting a variable and using it in the same statement is that the default result type of a variable is based on the type of the variable at the start of the statement. The following example illustrates this:

```
mysql> SET @a='test';
mysql> SELECT @a, (@a:=20) FROM tbl_name;
```

For this **SELECT** statement, MySQL reports to the client that column one is a string and converts all accesses of **@a** to strings, even though **@a** is set to a number for the second row. After the **SELECT** statement executes, **@a** is regarded as a number for the next statement.

To avoid problems with this behavior, either do not set and use the same variable within a single statement, or else set the variable to **0**, **0.0**, or **' '** to define its type before you use it.

A user variable cannot be used directly in an SQL statement as an identifier or as part of an identifier, even if it is set off with backticks. This is shown in the following example:

```
mysql> SELECT c1 FROM t;
+----+
| c1 |
+----+
| 0 |
+----+
| 1 |
+----+
2 rows in set (0.00 sec)

mysql> SET @col = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+-----+
| @col |
+-----+
| c1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT `@col` FROM t;
ERROR 1054 (42S22): UNKNOWN COLUMN '@COL' IN 'FIELD LIST'

mysql> SET @col = "`c1`";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+-----+
| @col |
+-----+
| `c1` |
+-----+
1 row in set (0.00 sec)
```

One way to work around this problem is to assemble a string for the query in application code, as shown here using PHP 5:

```
<?php
$mysqli = new mysqli("localhost", "user", "pass", "test");
```

```

if( mysqli_connect_errno() )
    die("Connection failed: %s\n", mysqli_connect_error());

$col = "c1";
$query = "SELECT $col FROM t";
$result = $mysqli->query($query);
while($row = $result->fetch_assoc())
{
    echo "<p>" . $row["$col"] . "</p>\n";
}

$result->close();
$mysqli->close();
?>

```

(Assembling an SQL statement in this fashion is sometimes known as “Dynamic SQL”). It is also possible to perform such operations using prepared statements, without the need to concatenate strings of SQL in client code. This example illustrates how this can be done:

```

mysql> SET @c = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SET @s = CONCAT("SELECT ", @c, " FROM t");
Query OK, 0 rows affected (0.00 sec)

mysql> PREPARE stmt FROM @s;
Query OK, 0 rows affected (0.04 sec)
Statement prepared

mysql> EXECUTE stmt;
+----+
| c1 |
+----+
| 0 |
+----+
| 1 |
+----+
2 rows in set (0.00 sec)

mysql> DEALLOCATE PREPARE stmt;
Query OK, 0 rows affected (0.00 sec)

```

You cannot use a placeholder for the name of a database, table, or column in an SQL prepared statement. See [Section 12.7, “SQL Syntax for Prepared Statements”](#), for more information.

8.5. Comment Syntax

MySQL Server supports three comment styles:

- From a “#” character to the end of the line.
- From a “-- ” sequence to the end of the line. In MySQL, the “-- ” (double-dash) comment style requires the second dash to be followed by at least one whitespace or control character (such as a space, tab, newline, and so on). This syntax differs slightly from standard SQL comment syntax, as discussed in [Section 1.9.5.6, “-- as the Start of a Comment”](#).
- From a /* sequence to the following */ sequence, as in the C programming language. This syntax allows a comment to extend over multiple lines because the beginning and closing sequences need not be on the same line.

The following example demonstrates all three comment styles:

```

mysql> SELECT 1+1;      # This comment continues to the end of line
mysql> SELECT 1+1;      -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;

```

MySQL Server supports some variants of C-style comments. These enable you to write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the “!” character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The comment syntax just described applies to how the `mysqld` server parses SQL statements. The `mysql` client program also performs some parsing of statements before sending them to the server. (It does this to determine statement boundaries within a multiple-statement input line.)

Chapter 9. Character Set Support

MySQL includes character set support that enables you to store data using a variety of character sets and perform comparisons according to a variety of collations. You can specify character sets at the server, database, table, and column level. MySQL supports the use of character sets for the [MyISAM](#), [MEMORY](#), [NDBCluster](#), and [InnoDB](#) storage engines.

This chapter discusses the following topics:

- What are character sets and collations?
- The multiple-level default system for character set assignment
- Syntax for specifying character sets and collations
- Affected functions and operations
- Unicode support
- The character sets and collations that are available, with notes

Character set issues affect data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the [utf8](#) Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about character set-related issues in client/server communication, see [Section 9.4, “Connection Character Sets and Collations”](#).

9.1. Character Sets and Collations in General

A *character set* is a set of symbols and encodings. A *collation* is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: 'A', 'B', 'a', 'b'. We give each letter a number: 'A' = 0, 'B' = 1, 'a' = 2, 'b' = 3. The letter 'A' is a symbol, the number 0 is the **encoding** for 'A', and the combination of all four letters and their encodings is a **character set**.

Suppose that we want to compare two string values, 'A' and 'B'. The simplest way to do this is to look at the encodings: 0 for 'A' and 1 for 'B'. Because 0 is less than 1, we say 'A' is less than 'B'. What we've just done is apply a collation to our character set. The collation is a set of rules (only one rule in this case): “compare the encodings.” We call this simplest of all possible collations a *binary* collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters 'a' and 'b' as equivalent to 'A' and 'B'; (2) then compare the encodings. We call this a *case-insensitive* collation. It's a little more complex than a binary collation.

In real life, most character sets have many characters: not just 'A' and 'B' but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with many special symbols and punctuation marks. Also in real life, most collations have many rules, not just for whether to distinguish lettercase, but also for whether to distinguish accents (an “accent” is a mark attached to a character as in German 'Ö'), and for multiple-character mappings (such as the rule that 'Ö' = 'OE' in one of the two German collations).

MySQL can do these things for you:

- Store strings using a variety of character sets
- Compare strings using a variety of collations
- Mix strings with different character sets or collations in the same server, the same database, or even the same table
- Allow specification of character set and collation at any level

In these respects, MySQL is far ahead of most other database management systems. However, to use these features effectively, you need to know what character sets and collations are available, how to change the defaults, and how they affect the behavior of string operators and functions.

9.2. Character Sets and Collations in MySQL

The MySQL server can support multiple character sets. To list the available character sets, use the `SHOW CHARACTER SET` statement. A partial listing follows. For more complete information, see [Section 9.11, “Character Sets and Collations That MySQL Supports”](#).

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
swe7	7bit Swedish	swe7_swedish_ci	1
ascii	US ASCII	ascii_general_ci	1
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci	1
tis620	TIS620 Thai	tis620_thai_ci	1
euckr	EUC-KR Korean	euckr_korean_ci	2
koi8u	KOI8-U Ukrainian	koi8u_general_ci	1
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
greek	ISO 8859-7 Greek	greek_general_ci	1
cp1250	Windows Central European	cp1250_general_ci	1
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	1

Any given character set always has at least one collation. It may have several collations. To list the collations for a character set, use the `SHOW COLLATION` statement. For example, to see the collations for the `latin1` (cp1252 West European) character set, use this statement to find those collation names that begin with `latin1`:

```
mysql> SHOW COLLATION LIKE 'latin1%';
```

Collation	Charset	Id	Default	Compiled	Sortlen
latin1_german1_ci	latin1	5			0
latin1_swedish_ci	latin1	8	Yes	Yes	1
latin1_danish_ci	latin1	15			0
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	1
latin1_general_ci	latin1	48			0
latin1_general_cs	latin1	49			0
latin1_spanish_ci	latin1	94			0

The `latin1` collations have the following meanings:

Collation	Meaning
<code>latin1_german1_ci</code>	German DIN-1
<code>latin1_swedish_ci</code>	Swedish/Finnish
<code>latin1_danish_ci</code>	Danish/Norwegian
<code>latin1_german2_ci</code>	German DIN-2
<code>latin1_bin</code>	Binary according to <code>latin1</code> encoding
<code>latin1_general_ci</code>	Multilingual (Western European)
<code>latin1_general_cs</code>	Multilingual (ISO Western European), case sensitive
<code>latin1_spanish_ci</code>	Modern Spanish

Collations have these general characteristics:

- Two different character sets cannot have the same collation.
- Each character set has one collation that is the *default collation*. For example, the default collation for `latin1` is `latin1_swedish_ci`. The output for `SHOW CHARACTER SET` indicates which collation is the default for each displayed character set.
- There is a convention for collation names: They start with the name of the character set with which they are associated, they usually include a language name, and they end with `_ci` (case insensitive), `_cs` (case sensitive), or `_bin` (binary).

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

9.3. Specifying Character Sets and Collations

There are default settings for character sets and collations at four levels: server, database, table, and column. The following description may appear complex, but it has been found in practice that multiple-level defaulting leads to natural and obvious results.

`CHARACTER SET` is used in clauses that specify a character set. `CHARSET` may be used as a synonym for `CHARACTER SET`.

9.3.1. Server Character Set and Collation

MySQL Server has a server character set and a server collation. These can be set at server startup on the command line or in an option file and changed at runtime.

Initially, the server character set and collation depend on the options that you use when you start `mysqld`. You can use `--character-set-server` for the character set. Along with it, you can add `--collation-server` for the collation. If you don't specify a character set, that is the same as saying `--character-set-server=latin1`. If you specify only a character set (for example, `latin1`) but not a collation, that is the same as saying `--character-set-server=latin1 --collation-server=latin1_swedish_ci` because `latin1_swedish_ci` is the default collation for `latin1`. Therefore, the following three commands all have the same effect:

```
shell> mysqld
shell> mysqld --character-set-server=latin1
shell> mysqld --character-set-server=latin1 \
       --collation-server=latin1_swedish_ci
```

One way to change the settings is by recompiling. If you want to change the default server character set and collation when building from sources, use: `--with-charset` and `--with-collation` as arguments for `configure`. For example:

```
shell> ./configure --with-charset=latin1
```

Or:

```
shell> ./configure --with-charset=latin1 \
       --with-collation=latin1_german1_ci
```

Both `mysqld` and `configure` verify that the character set/collation combination is valid. If not, each program displays an error message and terminates.

The current server character set and collation can be determined from the values of the `character_set_server` and `collation_server` system variables. These variables can be changed at runtime.

9.3.2. Database Character Set and Collation

Every database has a database character set and a database collation. The `CREATE DATABASE` and `ALTER DATABASE` statements have optional clauses for specifying the database character set and collation:

```
CREATE DATABASE db_name
  [[DEFAULT] CHARACTER SET charset_name
  [[DEFAULT] COLLATE collation_name

ALTER DATABASE db_name
  [[DEFAULT] CHARACTER SET charset_name
  [[DEFAULT] COLLATE collation_name
```

The keyword **SCHEMA** can be used instead of **DATABASE**.

All database options are stored in a text file named **db.opt** that can be found in the database directory.

The **CHARACTER SET** and **COLLATE** clauses make it possible to create databases with different character sets and collations on the same MySQL server.

Example:

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL chooses the database character set and database collation in the following manner:

- If both **CHARACTER SET *X*** and **COLLATE *Y*** were specified, then character set *X* and collation *Y*.
- If **CHARACTER SET *X*** was specified without **COLLATE**, then character set *X* and its default collation.
- If **COLLATE *Y*** was specified without **CHARACTER SET**, then the character set associated with *Y* and collation *Y*.
- Otherwise, the server character set and server collation.

The database character set and collation are used as default values if the table character set and collation are not specified in **CREATE TABLE** statements. They have no other purpose.

The character set and collation for the default database can be determined from the values of the **character_set_database** and **collation_database** system variables. The server sets these variables whenever the default database changes. If there is no default database, the variables have the same value as the corresponding server-level system variables, **character_set_server** and **collation_server**.

9.3.3. Table Character Set and Collation

Every table has a table character set and a table collation. The **CREATE TABLE** and **ALTER TABLE** statements have optional clauses for specifying the table character set and collation:

```
CREATE TABLE tbl_name (column_list)  
  [[DEFAULT] CHARACTER SET charset_name] [COLLATE collation_name]  
  
ALTER TABLE tbl_name  
  [[DEFAULT] CHARACTER SET charset_name] [COLLATE collation_name]
```

Example:

```
CREATE TABLE t1 ( ... ) CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL chooses the table character set and collation in the following manner:

- If both **CHARACTER SET *X*** and **COLLATE *Y*** were specified, then character set *X* and collation *Y*.
- If **CHARACTER SET *X*** was specified without **COLLATE**, then character set *X* and its default collation.
- If **COLLATE *Y*** was specified without **CHARACTER SET**, then the character set associated with *Y* and collation *Y*.
- Otherwise, the database character set and collation.

The table character set and collation are used as default values if the column character set and collation are not specified in individual column definitions. The table character set and collation are MySQL extensions; there are no such things in standard SQL.

9.3.4. Column Character Set and Collation

Every “character” column (that is, a column of type **CHAR**, **VARCHAR**, or **TEXT**) has a column character set and a column collation.

Column definition syntax has optional clauses for specifying the column character set and collation:

```
col_name {CHAR | VARCHAR | TEXT} (col_length)  
[CHARACTER SET charset_name] [COLLATE collation_name]
```

Example:

```
CREATE TABLE Table1  
(  
  column1 VARCHAR(5) CHARACTER SET latin1 COLLATE latin1_german1_ci  
);
```

MySQL chooses the column character set and collation in the following manner:

- If both **CHARACTER SET** *X* and **COLLATE** *Y* were specified, then character set *X* and collation *Y* are used.
- If **CHARACTER SET** *X* was specified without **COLLATE**, then character set *X* and its default collation are used.
- If **COLLATE** *Y* was specified without **CHARACTER SET**, then the character set associated with *Y* and collation *Y*.
- Otherwise, the table character set and collation are used.

The **CHARACTER SET** and **COLLATE** clauses are standard SQL.

9.3.5. Character String Literal Character Set and Collation

Every character string literal has a character set and a collation.

A character string literal may have an optional character set introducer and **COLLATE** clause:

```
[_charset_name] 'string' [COLLATE collation_name]
```

Examples:

```
SELECT 'string';  
SELECT _latin1'string';  
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

For the simple statement **SELECT** '*string*', the string has the character set and collation defined by the **character_set_connection** and **collation_connection** system variables.

The *_charset_name* expression is formally called an *introducer*. It tells the parser, “the string that is about to follow uses character set *X*.” Because this has confused people in the past, we emphasize that an introducer does not change the string to the introducer character set like **CONVERT()** would do. It does not change the string’s value, although padding may occur. The introducer is just a signal. An introducer is also legal before standard hex literal and numeric hex literal notation (*x'literal'* and *0xnnnn*)>.

Examples:

```
SELECT _latin1 x'AABBCC';  
SELECT _latin1 0xAABBCC;
```

MySQL determines a literal’s character set and collation in the following manner:

- If both *_X* and **COLLATE** *Y* were specified, then character set *X* and collation *Y* are used.
- If *_X* is specified but **COLLATE** is not specified, then character set *X* and its default collation are used.
- Otherwise, the character set and collation given by the **character_set_connection** and **collation_connection** system variables are used.

Examples:

- A string with `latin1` character set and `latin1_german1_ci` collation:

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
```

- A string with `latin1` character set and its default collation (that is, `latin1_swedish_ci`):

```
SELECT _latin1'Müller';
```

- A string with the connection default character set and collation:

```
SELECT 'Müller';
```

Character set introducers and the `COLLATE` clause are implemented according to standard SQL specifications.

An introducer indicates the character set for the following string, but does not change now how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by `character_set_connection`.

The following examples show that escape processing occurs using `character_set_connection` even in the presence of an introducer. The examples use `SET NAMES` (which changes `character_set_connection`, as discussed in [Section 9.4, “Connection Character Sets and Collations”](#)), and display the resulting strings using the `HEX()` function so that the exact string contents can be seen.

Example 1:

```
mysql> SET NAMES latin1;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX('à\n'), HEX(_sjis'à\n');
+-----+-----+
| HEX('à\n') | HEX(_sjis'à\n') |
+-----+-----+
| E00A      | E00A             |
+-----+-----+
1 row in set (0.00 sec)
```

Here, ‘à’ (hex value `E0`) is followed by ‘\n’, the escape sequence for newline. The escape sequence is interpreted using the `character_set_connection` value of `latin1` to produce a literal newline (hex value `0A`). This happens even for the second string. That is, the introducer of `_sjis` does not affect the parser’s escape processing.

Example 2:

```
mysql> SET NAMES sjis;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT HEX('à\n'), HEX(_latin1'à\n');
+-----+-----+
| HEX('à\n') | HEX(_latin1'à\n') |
+-----+-----+
| E05C6E     | E05C6E             |
+-----+-----+
1 row in set (0.04 sec)
```

Here, `character_set_connection` is `sjis`, a character set in which the sequence of ‘à’ followed by ‘\’ (hex values `05` and `5C`) is a valid multi-byte character. Hence, the first two bytes of the string are interpreted as a single `sjis` character, and the ‘\’ is not interpreted as an escape character. The following ‘n’ (hex value `6E`) is not interpreted as part of an escape sequence. This is true even for the second string; the introducer of `_latin1` does not affect escape processing.

9.3.6. National Character Set

Standard SQL defines `NCHAR` or `NATIONAL CHAR` as a way to indicate that a `CHAR` column should use some predefined character set. MySQL 5.1 uses `utf8` as this predefined character set. For example, these data type declarations are equivalent:

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

As are these:

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

You can use `N'literal'` (or `n'literal'`) to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For information on upgrading character sets to MySQL 5.1 from versions prior to 4.1, see the *MySQL 3.23, 4.0, 4.1 Reference Manual*.

9.3.7. Examples of Character Set and Collation Assignment

The following examples show how MySQL determines default character set and collation values.

Example 1: Table and Column Definition

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Here we have a column with a `latin1` character set and a `latin1_german1_ci` collation. The definition is explicit, so that's straightforward. Notice that there is no problem with storing a `latin1` column in a `latin2` table.

Example 2: Table and Column Definition

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

This time we have a column with a `latin1` character set and a default collation. Although it might seem natural, the default collation is not taken from the table level. Instead, because the default collation for `latin1` is always `latin1_swedish_ci`, column `c1` has a collation of `latin1_swedish_ci` (not `latin1_danish_ci`).

Example 3: Table and Column Definition

```
CREATE TABLE t1
(
  c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

We have a column with a default character set and a default collation. In this circumstance, MySQL checks the table level to determine the column character set and collation. Consequently, the character set for column `c1` is `latin1` and its collation is `latin1_danish_ci`.

Example 4: Database, Table, and Column Definition

```
CREATE DATABASE d1
  DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
  c1 CHAR(10)
);
```

We create a column without specifying its character set and collation. We're also not specifying a character set and a collation at the table level. In this circumstance, MySQL checks the database level to determine the table settings, which thereafter become the column settings.) Consequently, the character set for column `c1` is `latin2` and its collation is `latin2_czech_ci`.

9.3.8. Compatibility with Other DBMSs

For MaxDB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

9.4. Connection Character Sets and Collations

Several character set and collation system variables relate to a client's interaction with the server. Some of these have been mentioned in earlier sections:

- The server character set and collation can be determined from the values of the `character_set_server` and `collation_server` system variables.
- The character set and collation of the default database can be determined from the values of the `character_set_database` and `collation_database` system variables.

Additional character set and collation system variables are involved in handling traffic for the connection between a client and the server. Every client has connection-related character set and collation system variables.

Consider what a “connection” is: It's what you make when you connect to the server. The client sends SQL statements, such as queries, over the connection to the server. The server sends responses, such as result sets, over the connection back to the client. This leads to several questions about character set and collation handling for client connections, each of which can be answered in terms of system variables:

- What character set is the statement in when it leaves the client?

The server takes the `character_set_client` system variable to be the character set in which statements are sent by the client.

- What character set should the server translate a statement to after receiving it?

For this, the server uses the `character_set_connection` and `collation_connection` system variables. It converts statements sent by the client from `character_set_client` to `character_set_connection` (except for string literals that have an introducer such as `_latin1` or `_utf8`). `collation_connection` is important for comparisons of literal strings. For comparisons of strings with column values, `collation_connection` does not matter because columns have their own collation, which has a higher collation precedence.

- What character set should the server translate to before shipping result sets or error messages back to the client?

The `character_set_results` system variable indicates the character set in which the server returns query results to the client. This includes result data such as column values, and result metadata such as column names.

You can fine-tune the settings for these variables, or you can depend on the defaults (in which case, you can skip the rest of this section).

There are two statements that affect the connection character sets:

```
SET NAMES 'charset_name'
SET CHARACTER SET charset_name
```

`SET NAMES` indicates what character set the client will use to send SQL statements to the server. Thus, `SET NAMES 'cp1251'` tells the server “future incoming messages from this client are in character set `cp1251`.” It also specifies the character set that the server should use for sending results back to the client. (For example, it indicates what character set to use for column values if you use a `SELECT` statement.)

A `SET NAMES 'x'` statement is equivalent to these three statements:

```
SET character_set_client = x;
SET character_set_results = x;
SET character_set_connection = x;
```

Setting `character_set_connection` to `x` also sets `collation_connection` to the default collation for `x`. It is not necessary to set that collation explicitly. To specify a particular collation for the character sets, use the optional `COLLATE` clause:

```
SET NAMES 'charset_name' COLLATE 'collation_name'
```

SET CHARACTER SET is similar to **SET NAMES** but sets `character_set_connection` and `collation_connection` to `character_set_database` and `collation_database`. A **SET CHARACTER SET *x*** statement is equivalent to these three statements:

```
SET character_set_client = x;
SET character_set_results = x;
SET collation_connection = @@collation_database;
```

Setting `collation_connection` also sets `character_set_connection` to the character set associated with the collation (equivalent to executing `SET character_set_connection = @@character_set_database`). It is not necessary to set `character_set_connection` explicitly.

When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and `character_set_connection` system variables. In effect, the server performs a **SET NAMES** operation using the character set name.

With the `mysql` client, it is not necessary to execute **SET NAMES** every time you start up if you want to use a character set different from the default. You can add the `--default-character-set` option setting to your `mysql` statement line, or in your option file. For example, the following option file setting changes the three character set variables set to `koi8r` each time you invoke `mysql`:

```
[mysql]
default-character-set=koi8r
```

If you are using the `mysql` client with auto-reconnect enabled (which is not recommended), it is preferable to use the `charset` command rather than **SET NAMES**. For example:

```
mysql> charset utf8
Charset changed
```

The `charset` command issues a **SET NAMES** statement, and also changes the default character set that is used if `mysql` reconnects after the connection has dropped.

Example: Suppose that `column1` is defined as `CHAR(5) CHARACTER SET latin2`. If you do not say **SET NAMES** or **SET CHARACTER SET**, then for `SELECT column1 FROM t`, the server sends back all the values for `column1` using the character set that the client specified when it connected. On the other hand, if you say **SET NAMES 'latin1'** or **SET CHARACTER SET latin1** before issuing the **SELECT** statement, the server converts the `latin2` values to `latin1` just before sending results back. Conversion may be lossy if there are characters that are not in both character sets.

If you do not want the server to perform any conversion of result sets, set `character_set_results` to `NULL`:

```
SET character_set_results = NULL;
```

Note

Currently, UCS-2 cannot be used as a client character set, which means that **SET NAMES 'ucs2'** does not work.

To see the values of the character set and collation system variables that apply to your connection, use these statements:

```
SHOW VARIABLES LIKE 'character_set%';
SHOW VARIABLES LIKE 'collation%';
```

You must also consider the environment within which your MySQL application executes. For example, if you will send statements using UTF-8 text taken from a file that you create in an editor, you should edit the file with the locale of your environment set to UTF-8 so that the file's encoding is correct and so that the operating system handles it correctly. For a script that executes in a Web environment, the script must handle the character encoding properly for its interaction with the MySQL server, and it must generate pages that correctly indicate the encoding so that browsers know how to display the content of the pages.

9.5. Collation Issues

The following sections discuss various aspects of character set collations.

9.5.1. Using **COLLATE** in SQL Statements

With the **COLLATE** clause, you can override whatever the default collation is for a comparison. **COLLATE** may be used in various parts of SQL statements. Here are some examples:

- With **ORDER BY**:

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- With **AS**:

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```

- With **GROUP BY**:

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- With aggregate functions:

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- With **DISTINCT**:

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- With **WHERE**:

```
SELECT *
FROM t1
WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

```
SELECT *
FROM t1
WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
```

- With **HAVING**:

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

9.5.2. **COLLATE** Clause Precedence

The **COLLATE** clause has high precedence (higher than `| |`), so the following two expressions are equivalent:

```
x | | y COLLATE z
x | | (y COLLATE z)
```

9.5.3. **BINARY** Operator

The **BINARY** operator casts the string following it to a binary string. This is an easy way to force a comparison to be done byte by byte rather than character by character. **BINARY** also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
-> 1
mysql> SELECT BINARY 'a' = 'A';
```

```
mysql> -> 0
mysql> SELECT 'a' = 'a ';
mysql> -> 1
mysql> SELECT BINARY 'a' = 'a ';
mysql> -> 0
```

BINARY *str* is shorthand for **CAST(str AS BINARY)**.

The **BINARY** attribute in character column definitions has a different effect. A character column defined with the **BINARY** attribute is assigned the binary collation of the column's character set. Every character set has a binary collation. For example, the binary collation for the **latin1** character set is **latin1_bin**, so if the table default character set is **latin1**, these two column definitions are equivalent:

```
CHAR(10) BINARY
CHAR(10) CHARACTER SET latin1 COLLATE latin1_bin
```

The effect of **BINARY** as a column attribute differs from its effect prior to MySQL 4.1. Formerly, **BINARY** resulted in a column that was treated as a binary string. A binary string is a string of bytes that has no character set or collation, which differs from a non-binary character string that has a binary collation. For both types of strings, comparisons are based on the numeric values of the string unit, but for non-binary strings the unit is the character and some character sets allow multi-byte characters. [Section 10.4.2, “The BINARY and VARBINARY Types”](#).

The use of **CHARACTER SET binary** in the definition of a **CHAR**, **VARCHAR**, or **TEXT** column causes the column to be treated as a binary data type. For example, the following pairs of definitions are equivalent:

```
CHAR(10) CHARACTER SET binary
BINARY(10)

VARCHAR(10) CHARACTER SET binary
VARBINARY(10)

TEXT CHARACTER SET binary
BLOB
```

9.5.4. Some Special Cases Where the Collation Determination Is Tricky

In the great majority of statements, it is obvious what collation MySQL uses to resolve a comparison operation. For example, in the following cases, it should be clear that the collation is the collation of column **x**:

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

However, when multiple operands are involved, there can be ambiguity. For example:

```
SELECT x FROM T WHERE x = 'Y';
```

Should this query use the collation of the column **x**, or of the string literal **'Y'**?

Standard SQL resolves such questions using what used to be called “coercibility” rules. Basically, this means: Both **x** and **'Y'** have collations, so which collation takes precedence? This can be difficult to resolve, but the following rules cover most situations:

- An explicit **COLLATE** clause has a coercibility of 0. (Not coercible at all.)
- The concatenation of two strings with different collations has a coercibility of 1.
- The collation of a column or a stored routine parameter or local variable has a coercibility of 2.
- A “system constant” (the string returned by functions such as **USER()** or **VERSION()**) has a coercibility of 3.
- A literal's collation has a coercibility of 4.
- **NULL** or an expression that is derived from **NULL** has a coercibility of 5.

The preceding coercibility values are current for MySQL 5.1.

Those rules resolve ambiguities in the following manner:

- Use the collation with the lowest coercibility value.
- If both sides have the same coercibility, then:
 - If both sides are Unicode, or both sides are not Unicode, it is an error.
 - If one of the sides has a Unicode character set, and another side has a non-Unicode character set, the side with Unicode character set wins, and automatic character set conversion is applied to the non-Unicode side. For example, the following statement will not return an error:

```
SELECT CONCAT(utf8_column, latin1_column) FROM t1;
```

It will return a result, and the character set of the result will be `utf8`. The collation of the result will be the collation of `utf8_column`. Values of `latin1_column` will be automatically converted to `utf8` before concatenating.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings.

Examples:

<code>column1 = 'A'</code>	Use collation of <code>column1</code>
<code>column1 = 'A' COLLATE x</code>	Use collation of <code>'A' COLLATE x</code>
<code>column1 COLLATE x = 'A' COLLATE y</code>	Error

The `COERCIBILITY()` function can be used to determine the coercibility of a string expression:

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
-> 0
mysql> SELECT COERCIBILITY(VERSION());
-> 3
mysql> SELECT COERCIBILITY('A');
-> 4
```

See [Section 11.11.3, “Information Functions”](#).

9.5.5. Collations Must Be for the Right Character Set

Each character set has one or more collations, but each collation is associated with one and only one character set. Therefore, the following statement causes an error message because the `latin2_bin` collation is not legal with the `latin1` character set:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET '_latin1'
```

9.5.6. Examples of the Effect of Collation

Example 1: Sorting German Umlauts

Suppose that column `X` in table `T` has these `latin1` column values:

```
Mufler
Müller
MX Systems
MySQL
```

Suppose also that the column values are retrieved using the following statement:

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```


The following table shows the resulting order of the values if we use `ORDER BY` with different collations:

<code>latin1_swedish_ci</code>	<code>latin1_german1_ci</code>	<code>latin1_german2_ci</code>
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

The character that causes the different sort orders in this example is the U with two dots over it (ü), which the Germans call “U-umlaut.”

- The first column shows the result of the `SELECT` using the Swedish/Finnish collating rule, which says that U-umlaut sorts with Y.
- The second column shows the result of the `SELECT` using the German DIN-1 rule, which says that U-umlaut sorts with U.
- The third column shows the result of the `SELECT` using the German DIN-2 rule, which says that U-umlaut sorts with UE.

Example 2: Searching for German Umlauts

Suppose that you have three tables that differ only by the character set and collation used:

```
mysql> CREATE TABLE german1 (
->   c CHAR(10)
-> ) CHARACTER SET latin1 COLLATE latin1_german1_ci;
mysql> CREATE TABLE german2 (
->   c CHAR(10)
-> ) CHARACTER SET latin1 COLLATE latin1_german2_ci;
mysql> CREATE TABLE germanutf8 (
->   c CHAR(10)
-> ) CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Each table contains two records:

```
mysql> INSERT INTO german1 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO german2 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO germanutf8 VALUES ('Bar'), ('Bär');
```

Two of the above collations have an `A = Ä` equality, and one has no such equality (`latin1_german2_ci`). For that reason, you'll get these results in comparisons:

```
mysql> SELECT * FROM german1 WHERE c = 'Bär';
+-----+
| c      |
+-----+
| Bar    |
| Bär    |
+-----+
mysql> SELECT * FROM german2 WHERE c = 'Bär';
+-----+
| c      |
+-----+
| Bär    |
+-----+
mysql> SELECT * FROM germanutf8 WHERE c = 'Bär';
+-----+
| c      |
+-----+
| Bar    |
| Bär    |
+-----+
```

This is not a bug but rather a consequence of the sorting that `latin1_german1_ci` or `utf8_unicode_ci` do (the sorting shown is done according to the German DIN 5007 standard).

9.6. String Repertoire

As of MySQL 5.1.21, string expressions have an attribute known as *repertoire*, which can have two values:

- **ASCII**: The expression can contain only characters in the Unicode range **U+0000** to **U+007F**.
- **UNICODE**: The expression can contain characters in the Unicode range **U+0000** to **U+FFFF**.

The **ASCII** range is a subset of **UNICODE** range, so a string with **ASCII** repertoire can be converted safely without loss of information to the character set of any string with **UNICODE** repertoire or to a character set that is a superset of **ASCII**. (All MySQL character sets are supersets of **ASCII** with the exception of **swe7**, which reuses some punctuation characters for Swedish accented characters.) The use of repertoire enables character set conversion in expressions for many cases where MySQL would otherwise return an “illegal mix of collations” error.

The following discussion provides examples of expressions and their repertoires, and describes how the use of repertoire changes string expression evaluation:

- The repertoire for string constants depends on string content:

```
SET NAMES utf8; SELECT 'abc';
SELECT _utf8'def';
SELECT N'MySQL';
```

Although the character set is **utf8** in each of the preceding cases, the strings do not actually contain any characters outside the **ASCII** range, so their repertoire is **ASCII** rather than **UNICODE**.

- Columns having the **ascii** character set have **ASCII** repertoire because of their character set. In the following table, **c1** has **ASCII** repertoire:

```
CREATE TABLE t1 (c1 CHAR(1) CHARACTER SET ascii);
```

The following example illustrates how repertoire enables a result to be determined in a case where an error occurs without repertoire:

```
CREATE TABLE t1 (
  c1 CHAR(1) CHARACTER SET latin1,
  c2 CHAR(1) CHARACTER SET ascii
);
INSERT INTO t1 VALUES ('a','b');
SELECT CONCAT(c1,c2) FROM t1;
```

Without repertoire, this error occurs:

```
ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
and (ascii_general_ci,IMPLICIT) for operation 'concat'
```

Using repertoire, subset to superset (**ascii** to **latin1**) conversion can occur and a result is returned:

```
+-----+
| CONCAT(c1,c2) |
+-----+
| ab            |
+-----+
```

- Functions with one string argument inherit the repertoire of their argument. The result of **UPPER(_utf8'abc')** has **ASCII** repertoire, because its argument has **ASCII** repertoire.
- For functions that return a string but do not have string arguments and use **character_set_connection** as the result character set, the result repertoire is **ASCII** if **character_set_connection** is **ascii**, and **UNICODE** otherwise:

```
FORMAT(numeric_column, 4);
```

Use of repertoire changes how MySQL evaluates the following example:

```
SET NAMES ascii;
CREATE TABLE t1 (a INT, b VARCHAR(10) CHARACTER SET latin1);
INSERT INTO t1 VALUES (1,'b');
```

```
SELECT CONCAT(FORMAT(a, 4), b) FROM t1;
```

Without repertoire, this error occurs:

```
ERROR 1267 (HY000): Illegal mix of collations (ascii_general_ci,COERCIBLE)
and (latin1_swedish_ci,IMPLICIT) for operation 'concat'
```

With repertoire, a result is returned:

```
+-----+
| CONCAT(FORMAT(a, 4), b) |
+-----+
| 1.0000b                 |
+-----+
```

- Functions with two or more string arguments use the “widest” argument repertoire for the result repertoire (**UNICODE** is wider than **ASCII**). Consider the following **CONCAT()** calls:

```
CONCAT(_ucs2 0x0041, _ucs2 0x0042)
CONCAT(_ucs2 0x0041, _ucs2 0x00C2)
```

For the first call, the repertoire is **ASCII** because both arguments are within the range of the **ascii** character set. For the second call, the repertoire is **UNICODE** because the second argument is outside the **ascii** character set range.

- The repertoire for function return values is determined based only on the repertoire of the arguments that affect the result's character set and collation.

```
IF(column1 < column2, 'smaller', 'greater')
```

The result repertoire is **ASCII** because the two string arguments (the second argument and the third argument) both have **ASCII** repertoire. The first argument does not matter for the result repertoire, even if the expression uses string values.

9.7. Operations Affected by Character Set Support

This section describes operations that take character set information into account.

9.7.1. Result Strings

MySQL has many operators and functions that return a string. This section answers the question: What is the character set and collation of such a string?

For simple functions that take string input and return a string result as output, the output's character set and collation are the same as those of the principal input value. For example, **UPPER(X)** returns a string whose character string and collation are the same as that of **X**. The same applies for **INSTR()**, **LCASE()**, **LOWER()**, **LTRIM()**, **MID()**, **REPEAT()**, **REPLACE()**, **REVERSE()**, **RIGHT()**, **RPAD()**, **RTRIM()**, **SOUNDEX()**, **SUBSTRING()**, **TRIM()**, **UCASE()**, and **UPPER()**.

Note: The **REPLACE()** function, unlike all other functions, always ignores the collation of the string input and performs a case-sensitive comparison.

If a string input or function result is a binary string, the string has no character set or collation. This can be checked by using the **CHARSET()** and **COLLATION()** functions, both of which return **binary** to indicate that their argument is a binary string:

```
mysql> SELECT CHARSET(BINARY 'a'), COLLATION(BINARY 'a');
+-----+-----+
| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |
+-----+-----+
| binary              | binary                |
+-----+-----+
```

For operations that combine multiple string inputs and return a single string output, the “aggregation rules” of standard SQL apply for determining the collation of the result:

- If an explicit `COLLATE X` occurs, use `X`.
- If explicit `COLLATE X` and `COLLATE Y` occur, raise an error.
- Otherwise, if all collations are `X`, use `X`.
- Otherwise, the result has no collation.

For example, with `CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END`, the resulting collation is `X`. The same applies for `UNION`, `||`, `CONCAT()`, `ELT()`, `GREATEST()`, `IF()`, and `LEAST()`.

For operations that convert to character data, the character set and collation of the strings that result from the operations are defined by the `character_set_connection` and `collation_connection` system variables. This applies only to `CAST()`, `CONV()`, `FORMAT()`, `HEX()`, and `SPACE()`.

If you are uncertain about the character set or collation of the result returned by a string function, you can use the `CHARSET()` or `COLLATE()` function to find out:

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+-----+-----+-----+
| USER()          | CHARSET(USER()) | COLLATION(USER()) |
+-----+-----+-----+
| test@localhost | utf8            | utf8_general_ci   |
+-----+-----+-----+
```

9.7.2. CONVERT() and CAST()

`CONVERT()` provides a way to convert data between different character sets. The syntax is:

```
CONVERT(expr USING transcoding_name)
```

In MySQL, transcoding names are the same as the corresponding character set names.

Examples:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
  SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

`CONVERT(... USING ...)` is implemented according to the standard SQL specification.

You may also use `CAST()` to convert a string to a different character set. The syntax is:

```
CAST(character_string AS character_data_type CHARACTER SET charset_name)
```

Example:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

If you use `CAST()` without specifying `CHARACTER SET`, the resulting character set and collation are defined by the `character_set_connection` and `collation_connection` system variables. If you use `CAST()` with `CHARACTER SET X`, the resulting character set and collation are `X` and the default collation of `X`.

You may not use a `COLLATE` clause inside a `CAST()`, but you may use it outside. That is, `CAST(... COLLATE ...)` is illegal, but `CAST(...) COLLATE ...` is legal.

Example:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

9.7.3. SHOW Statements and INFORMATION_SCHEMA

Several `SHOW` statements provide additional character set information. These include `SHOW CHARACTER SET`, `SHOW COLLATION`,

`SHOW CREATE DATABASE`, `SHOW CREATE TABLE` and `SHOW COLUMNS`. These statements are described here briefly.

For more information, see [Section 12.5.4, “SHOW Syntax”](#).

`INFORMATION_SCHEMA` has several tables that contain information similar to that displayed by the `SHOW` statements. For example, the `CHARACTER_SETS` and `COLLATIONS` tables contain the information displayed by `SHOW CHARACTER SET` and `SHOW COLLATION`. [Chapter 23, The INFORMATION_SCHEMA Database](#).

The `SHOW CHARACTER SET` command shows all available character sets. It takes an optional `LIKE` clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
```

Charset	Description	Default collation	Maxlen
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	1
latin7	ISO 8859-13 Baltic	latin7_general_ci	1

The output from `SHOW COLLATION` includes all available character sets. It takes an optional `LIKE` clause that indicates which collation names to match. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
```

Collation	Charset	Id	Default	Compiled	Sortlen
latin1_german1_ci	latin1	5			0
latin1_swedish_ci	latin1	8	Yes	Yes	0
latin1_danish_ci	latin1	15			0
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	0
latin1_general_ci	latin1	48			0
latin1_general_cs	latin1	49			0
latin1_spanish_ci	latin1	94			0

`SHOW CREATE DATABASE` displays the `CREATE DATABASE` statement that creates a given database:

```
mysql> SHOW CREATE DATABASE test;
```

Database	Create Database
test	CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */

If no `COLLATE` clause is shown, the default collation for the character set applies.

`SHOW CREATE TABLE` is similar, but displays the `CREATE TABLE` statement to create a given table. The column definitions indicate any character set specifications, and the table options include character set information.

The `SHOW COLUMNS` statement displays the collations of a table's columns when invoked as `SHOW FULL COLUMNS`. Columns with `CHAR`, `VARCHAR`, or `TEXT` data types have collations. Numeric and other non-character types have no collation (indicated by `NULL` as the `Collation` value). For example:

```
mysql> SHOW FULL COLUMNS FROM person\G
***** 1. row *****
Field: id
Type: smallint(5) unsigned
Collation: NULL
Null: NO
Key: PRI
Default: NULL
Extra: auto_increment
Privileges: select,insert,update,references
Comment:
***** 2. row *****
Field: name
Type: char(60)
Collation: latin1_swedish_ci
Null: NO
Key:
Default:
Extra:
Privileges: select,insert,update,references
Comment:
```

The character set is not part of the display but is implied by the collation name.

9.8. Unicode Support

MySQL 5.1 supports two character sets for storing Unicode data:

- `ucs2`, the UCS-2 Unicode character set.
- `utf8`, the UTF-8 encoding of the Unicode character set.

In UCS-2 (binary Unicode representation), every character is represented by a two-byte Unicode code with the most significant byte first. For example: `LATIN CAPITAL LETTER A` has the code `0x0041` and it is stored as a two-byte sequence: `0x00 0x41`. `CYRILLIC SMALL LETTER YERU` (Unicode `0x044B`) is stored as a two-byte sequence: `0x04 0x4B`. For Unicode characters and their codes, please refer to the [Unicode Home Page](#).

The MySQL implementation of UCS-2 stores characters in big-endian byte order and does not use a byte order mark (BOM) at the beginning of UCS-2 values. Other database systems might use little-ending byte order or a BOM, in which case conversion of UCS-2 values will need to be performed when transferring data between those systems and MySQL.

Currently, UCS-2 cannot be used as a client character set, which means that `SET NAMES 'ucs2'` does not work.

UTF-8 (Unicode Transformation Format with 8-bit units) is an alternative way to store Unicode data. It is implemented according to RFC 3629. The idea of UTF-8 is that various Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and Middle East script letters fit into a two-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.
- Korean, Chinese, and Japanese ideographs use three-byte sequences.

RFC 3629 describes encoding sequences that take from one to four bytes. Currently, MySQL support for UTF-8 does not include four-byte sequences. (An older standard for UTF-8 encoding is given by RFC 2279, which describes UTF-8 sequences that take from one to six bytes. RFC 3629 renders RFC 2279 obsolete; for this reason, sequences with five and six bytes are no longer used.)

MySQL uses no BOM for UTF-8 values.

Tip: To save space with UTF-8, use `VARCHAR` instead of `CHAR`. Otherwise, MySQL must reserve three bytes for each character in a `CHAR CHARACTER SET utf8` column because that is the maximum possible length. For example, MySQL must reserve 30 bytes for a `CHAR(10) CHARACTER SET utf8` column.

9.9. UTF-8 for Metadata

Metadata is “the data about the data.” Anything that *describes* the database — as opposed to being the *contents* of the database — is metadata. Thus column names, database names, usernames, version names, and most of the string results from `SHOW` are metadata. This is also true of the contents of tables in `INFORMATION_SCHEMA`, because those tables by definition contain information about database objects.

Representation of metadata must satisfy these requirements:

- All metadata must be in the same character set. Otherwise, neither the `SHOW` commands nor `SELECT` statements for tables in `INFORMATION_SCHEMA` would work properly because different rows in the same column of the results of these operations would be in different character sets.
- Metadata must include all characters in all languages. Otherwise, users would not be able to name columns and tables using their own languages.

To satisfy both requirements, MySQL stores metadata in a Unicode character set, namely UTF-8. This does not cause any disruption if

you never use accented or non-Latin characters. But if you do, you should be aware that metadata is in UTF-8.

The metadata requirements mean that the return values of the `USER()`, `CURRENT_USER()`, `SESSION_USER()`, `SYSTEM_USER()`, `DATABASE()`, and `VERSION()` functions have the UTF-8 character set by default.

The server sets the `character_set_system` system variable to the name of the metadata character set:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_system | utf8 |
+-----+-----+
```

Storage of metadata using Unicode does *not* mean that the server returns headers of columns and the results of `DESCRIBE` functions in the `character_set_system` character set by default. When you use `SELECT column1 FROM t`, the name `column1` itself is returned from the server to the client in the character set determined by the value of the `character_set_results` system variable, which has a default value of `latin1`. If you want the server to pass metadata results back in a different character set, use the `SET NAMES` statement to force the server to perform character set conversion. `SET NAMES` sets the `character_set_results` and other related system variables. (See [Section 9.4, “Connection Character Sets and Collations”](#).) Alternatively, a client program can perform the conversion after receiving the result from the server. It is more efficient for the client perform the conversion, but this option is not always available for all clients.

If `character_set_results` is set to `NULL`, no conversion is performed and the server returns metadata using its original character set (the set indicated by `character_set_system`).

Error messages returned from the server to the client are converted to the client character set automatically, as with metadata.

If you are using (for example) the `USER()` function for comparison or assignment within a single statement, don't worry. MySQL performs some automatic conversion for you.

```
SELECT * FROM Table1 WHERE USER() = latin1_column;
```

This works because the contents of `latin1_column` are automatically converted to UTF-8 before the comparison.

```
INSERT INTO Table1 (latin1_column) SELECT USER();
```

This works because the contents of `USER()` are automatically converted to `latin1` before the assignment. Automatic conversion is not fully implemented yet, but should work correctly in a later version.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings. For more information about coercion of strings, see [Section 9.5.4, “Some Special Cases Where the Collation Determination Is Tricky”](#).

9.10. Column Character Set Conversion

To convert a binary or non-binary string column to use a particular character set, use `ALTER TABLE`. For successful conversion to occur, one of the following conditions must apply:

- If the column has a binary data type (`BINARY`, `VARBINARY`, `BLOB`), all the values that it contains must be encoded using a single character set (the character set you're converting the column to). If you use a binary column to store information in multiple character sets, MySQL has no way to know which values use which character set and cannot convert the data properly.
- If the column has a non-binary data type (`CHAR`, `VARCHAR`, `TEXT`), its contents should be encoded in the column's character set, not some other character set. If the contents are encoded in a different character set, you can convert the column to use a binary data type first, and then to a non-binary column with the desired character set.

Suppose that a table `t` has a binary column named `col1` defined as `BINARY(50)`. Assuming that the information in the column is encoded using a single character set, you can convert it to a non-binary column that has that character set. For example, if `col1` contains binary data representing characters in the `greek` character set, you can convert it as follows:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET greek;
```

Suppose that table `t` has a non-binary column named `col1` defined as `CHAR(50) CHARACTER SET latin1` but you want to convert it to use `utf8` so that you can store values from many languages. The following statement accomplishes this:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET utf8;
```

Conversion may be lossy if the column contains characters that are not in both character sets.

A special case occurs if you have old tables from MySQL 4.0 or earlier where a non-binary column contains values that actually are encoded in a character set different from the server's default character set. For example, an application might have stored `sjis` values in a column, even though MySQL's default character set was `latin1`. It is possible to convert the column to use the proper character set but an additional step is required. Suppose that the server's default character set was `latin1` and `col1` is defined as `CHAR(50)` but its contents are `sjis` values. The first step is to convert the column to a binary data type, which removes the existing character set information without performing any character conversion:

```
ALTER TABLE t MODIFY col1 BINARY(50);
```

The next step is to convert the column to a non-binary data type with the proper character set:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET sjis;
```

This procedure requires that the table not have been modified already with statements such as `INSERT` or `UPDATE` after an upgrade to MySQL 4.1 or later. In that case, MySQL would store new values in the column using `latin1`, and the column will contain a mix of `sjis` and `latin1` values and cannot be converted properly.

If you specified attributes when creating a column initially, you should also specify them when altering the table with `ALTER TABLE`. For example, if you specified `NOT NULL` and an explicit `DEFAULT` value, you should also provide them in the `ALTER TABLE` statement. Otherwise, the resulting column definition will not include those attributes.

9.11. Character Sets and Collations That MySQL Supports

MySQL supports 70+ collations for 30+ character sets. This section indicates which character sets MySQL supports. There is one subsection for each group of related character sets. For each character set, the allowable collations are listed.

You can always list the available character sets and their default collations with the `SHOW CHARACTER SET` statement:

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation
big5	Big5 Traditional Chinese	big5_chinese_ci
dec8	DEC West European	dec8_swedish_ci
cp850	DOS West European	cp850_general_ci
hp8	HP West European	hp8_english_ci
koi8r	KOI8-R Relcom Russian	koi8r_general_ci
latin1	cp1252 West European	latin1_swedish_ci
latin2	ISO 8859-2 Central European	latin2_general_ci
swe7	7bit Swedish	swe7_swedish_ci
ascii	US ASCII	ascii_general_ci
ujis	EUC-JP Japanese	ujis_japanese_ci
sjis	Shift-JIS Japanese	sjis_japanese_ci
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci
tis620	TIS620 Thai	tis620_thai_ci
euckr	EUC-KR Korean	euckr_korean_ci
koi8u	KOI8-U Ukrainian	koi8u_general_ci
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci
greek	ISO 8859-7 Greek	greek_general_ci
cp1250	Windows Central European	cp1250_general_ci
gbk	GBK Simplified Chinese	gbk_chinese_ci
latin5	ISO 8859-9 Turkish	latin5_turkish_ci
armscii8	ARMSCII-8 Armenian	armscii8_general_ci
utf8	UTF-8 Unicode	utf8_general_ci
ucs2	UCS-2 Unicode	ucs2_general_ci
cp866	DOS Russian	cp866_general_ci
keybcs2	DOS Kamenicky Czech-Slovak	keybcs2_general_ci
macce	Mac Central European	macce_general_ci
macroman	Mac West European	macroman_general_ci
cp852	DOS Central European	cp852_general_ci
latin7	ISO 8859-13 Baltic	latin7_general_ci
cp1251	Windows Cyrillic	cp1251_general_ci
cp1256	Windows Arabic	cp1256_general_ci
cp1257	Windows Baltic	cp1257_general_ci
binary	Binary pseudo charset	binary
geostd8	GEOSTD8 Georgian	geostd8_general_ci
cp932	SJIS for Windows Japanese	cp932_japanese_ci


```
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |  
+-----+-----+-----+
```

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

9.11.1. Unicode Character Sets

MySQL has two Unicode character sets. You can store text in about 650 languages using these character sets.

- **ucs2** (UCS-2 Unicode) collations:

- `ucs2_bin`
- `ucs2_czech_ci`
- `ucs2_danish_ci`
- `ucs2_esperanto_ci`
- `ucs2_estonian_ci`
- `ucs2_general_ci` (default)
- `ucs2_hungarian_ci`
- `ucs2_icelandic_ci`
- `ucs2_latvian_ci`
- `ucs2_lithuanian_ci`
- `ucs2_persian_ci`
- `ucs2_polish_ci`
- `ucs2_roman_ci`
- `ucs2_romanian_ci`
- `ucs2_slovak_ci`
- `ucs2_slovenian_ci`
- `ucs2_spanish2_ci`
- `ucs2_spanish_ci`
- `ucs2_swedish_ci`
- `ucs2_turkish_ci`
- `ucs2_unicode_ci`

- **utf8** (UTF-8 Unicode) collations:

- `utf8_bin`
- `utf8_czech_ci`
- `utf8_danish_ci`
- `utf8_esperanto_ci`
- `utf8_estonian_ci`

- `utf8_general_ci` (default)
- `utf8_hungarian_ci`
- `utf8_iceLandic_ci`
- `utf8_latvian_ci`
- `utf8_lithuanian_ci`
- `utf8_persian_ci`
- `utf8_polish_ci`
- `utf8_roman_ci`
- `utf8_romanian_ci`
- `utf8_slovak_ci`
- `utf8_slovenian_ci`
- `utf8_spanish2_ci`
- `utf8_spanish_ci`
- `utf8_swedish_ci`
- `utf8_turkish_ci`
- `utf8_unicode_ci`

The MySQL implementation of UCS-2 stores characters in big-endian byte order and does not use a byte order mark (BOM) at the beginning of UCS-2 values. Other database systems might use little-ending byte order or a BOM, in which case conversion of UCS-2 values will need to be performed when transferring data between those systems and MySQL.

Note that in the `ucs2_roman_ci` and `utf8_roman_ci` collations, **I** and **J** compare as equals, and **U** and **V** compare as equals.

The `ucs2_hungarian_ci` and `utf8_hungarian_ci` collations were added in MySQL 5.1.5.

MySQL implements the `utf8_unicode_ci` collation according to the Unicode Collation Algorithm (UCA) described at <http://www.unicode.org/reports/tr10/>. The collation uses the version-4.0.0 UCA weight keys: <http://www.unicode.org/Public/UC4.0.0/allkeys-4.0.0.txt>. The following discussion uses `utf8_unicode_ci`, but it is also true for `ucs2_unicode_ci`.

Currently, the `utf8_unicode_ci` collation has only partial support for the Unicode Collation Algorithm. Some characters are not supported yet. Also, combining marks are not fully supported. This affects primarily Vietnamese, Yoruba, and some smaller languages such as Navajo.

The most significant feature in `utf8_unicode_ci` is that it supports expansions; that is, when one character compares as equal to combinations of other characters. For example, in German and some other languages ‘ß’ is equal to ‘ss’.

`utf8_general_ci` is a legacy collation that does not support expansions. It can make only one-to-one comparisons between characters. This means that comparisons for the `utf8_general_ci` collation are faster, but slightly less correct, than comparisons for `utf8_unicode_ci`.

For example, the following equalities hold in both `utf8_general_ci` and `utf8_unicode_ci` (for the effect this has in comparisons or when doing searches, see [Section 9.5.6, “Examples of the Effect of Collation”](#)):

```
Ä = A
Ö = O
Ü = U
```

A difference between the collations is that this is true for `utf8_general_ci`:

```
ß = s
```

Whereas this is true for `utf8_unicode_ci`:

```
ß = ss
```

MySQL implements language-specific collations for the `utf8` character set only if the ordering with `utf8_unicode_ci` does not work well for a language. For example, `utf8_unicode_ci` works fine for German and French, so there is no need to create special `utf8` collations for these two languages.

`utf8_general_ci` also is satisfactory for both German and French, except that ‘ß’ is equal to ‘s’, and not to ‘ss’. If this is acceptable for your application, then you should use `utf8_general_ci` because it is faster. Otherwise, use `utf8_unicode_ci` because it is more accurate.

`utf8_swedish_ci`, like other `utf8` language-specific collations, is derived from `utf8_unicode_ci` with additional language rules. For example, in Swedish, the following relationship holds, which is not something expected by a German or French speaker:

```
Ü = Υ < Ö
```

The `utf8_spanish_ci` and `utf8_spanish2_ci` collations correspond to modern Spanish and traditional Spanish, respectively. In both collations, ‘ñ’ (n-tilde) is a separate letter between ‘n’ and ‘o’. In addition, for traditional Spanish, ‘ch’ is a separate letter between ‘c’ and ‘d’, and ‘ll’ is a separate letter between ‘l’ and ‘m’.

9.11.2. West European Character Sets

Western European character sets cover most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English.

- `ascii` (US ASCII) collations:
 - `ascii_bin`
 - `ascii_general_ci` (default)
- `cp850` (DOS West European) collations:
 - `cp850_bin`
 - `cp850_general_ci` (default)
- `dec8` (DEC Western European) collations:
 - `dec8_bin`
 - `dec8_swedish_ci` (default)
- `hp8` (HP Western European) collations:
 - `hp8_bin`
 - `hp8_english_ci` (default)
- `latin1` (cp1252 West European) collations:
 - `latin1_bin`
 - `latin1_danish_ci`
 - `latin1_general_ci`
 - `latin1_general_cs`
 - `latin1_german1_ci`

- `latin1_german2_ci`
- `latin1_spanish_ci`
- `latin1_swedish_ci` (default)

`latin1` is the default character set. MySQL's `latin1` is the same as the Windows `cp1252` character set. This means it is the same as the official [ISO 8859-1](#) or IANA (Internet Assigned Numbers Authority) `latin1`, except that IANA `latin1` treats the code points between `0x80` and `0x9f` as “undefined,” whereas `cp1252`, and therefore MySQL's `latin1`, assign characters for those positions. For example, `0x80` is the Euro sign. For the “undefined” entries in `cp1252`, MySQL translates `0x81` to Unicode `0x0081`, `0x8d` to `0x008d`, `0x8f` to `0x008f`, `0x90` to `0x0090`, and `0x9d` to `0x009d`.

The `latin1_swedish_ci` collation is the default that probably is used by the majority of MySQL customers. Although it is frequently said that it is based on the Swedish/Finnish collation rules, there are Swedes and Finns who disagree with this statement.

The `latin1_german1_ci` and `latin1_german2_ci` collations are based on the DIN-1 and DIN-2 standards, where DIN stands for *Deutsches Institut für Normung* (the German equivalent of ANSI). DIN-1 is called the “dictionary collation” and DIN-2 is called the “phone book collation.” For an example of the effect this has in comparisons or when doing searches, see [Section 9.5.6, “Examples of the Effect of Collation”](#).

- `latin1_german1_ci` (dictionary) rules:

```
Ä = A
Ö = O
Ü = U
ß = s
```

- `latin1_german2_ci` (phone-book) rules:

```
Ä = AE
Ö = OE
Ü = UE
ß = SS
```

For an example of the effect this has in comparisons or when doing searches, see [Section 9.5.6, “Examples of the Effect of Collation”](#).

In the `latin1_spanish_ci` collation, ‘ñ’ (n-tilde) is a separate letter between ‘n’ and ‘o’.

- `macroman` (Mac West European) collations:
 - `macroman_bin`
 - `macroman_general_ci` (default)
- `swe7` (7bit Swedish) collations:
 - `swe7_bin`
 - `swe7_swedish_ci` (default)

9.11.3. Central European Character Sets

MySQL provides some support for character sets used in the Czech Republic, Slovakia, Hungary, Romania, Slovenia, Croatia, Poland, and Serbia (Latin).

- `cp1250` (Windows Central European) collations:
 - `cp1250_bin`
 - `cp1250_croatian_ci`
 - `cp1250_czech_cs`

- `cp1250_general_ci` (default)
- `cp1250_polish_ci`
- `cp852` (DOS Central European) collations:
 - `cp852_bin`
 - `cp852_general_ci` (default)
- `keybcs2` (DOS Kamenicky Czech-Slovak) collations:
 - `keybcs2_bin`
 - `keybcs2_general_ci` (default)
- `latin2` (ISO 8859-2 Central European) collations:
 - `latin2_bin`
 - `latin2_croatian_ci`
 - `latin2_czech_cs`
 - `latin2_general_ci` (default)
 - `latin2_hungarian_ci`
- `macce` (Mac Central European) collations:
 - `macce_bin`
 - `macce_general_ci` (default)

9.11.4. South European and Middle East Character Sets

South European and Middle Eastern character sets supported by MySQL include Armenian, Arabic, Georgian, Greek, Hebrew, and Turkish.

- `armscii8` (ARMScii8 Armenian) collations:
 - `armscii8_bin`
 - `armscii8_general_ci` (default)
- `cp1256` (Windows Arabic) collations:
 - `cp1256_bin`
 - `cp1256_general_ci` (default)
- `geostd8` (GEOSTD8 Georgian) collations:
 - `geostd8_bin`
 - `geostd8_general_ci` (default)
- `greek` (ISO 8859-7 Greek) collations:
 - `greek_bin`
 - `greek_general_ci` (default)

- [hebrew](#) (ISO 8859-8 Hebrew) collations:
 - [hebrew_bin](#)
 - [hebrew_general_ci](#) (default)
- [latin5](#) (ISO 8859-9 Turkish) collations:
 - [latin5_bin](#)
 - [latin5_turkish_ci](#) (default)

9.11.5. Baltic Character Sets

The Baltic character sets cover Estonian, Latvian, and Lithuanian languages.

- [cp1257](#) (Windows Baltic) collations:
 - [cp1257_bin](#)
 - [cp1257_general_ci](#) (default)
 - [cp1257_lithuanian_ci](#)
- [latin7](#) (ISO 8859-13 Baltic) collations:
 - [latin7_bin](#)
 - [latin7_estonian_cs](#)
 - [latin7_general_ci](#) (default)
 - [latin7_general_cs](#)

9.11.6. Cyrillic Character Sets

The Cyrillic character sets and collations are for use with Belarusian, Bulgarian, Russian, Ukrainian, and Serbian (Cyrillic) languages.

- [cp1251](#) (Windows Cyrillic) collations:
 - [cp1251_bin](#)
 - [cp1251_bulgarian_ci](#)
 - [cp1251_general_ci](#) (default)
 - [cp1251_general_cs](#)
 - [cp1251_ukrainian_ci](#)
- [cp866](#) (DOS Russian) collations:
 - [cp866_bin](#)
 - [cp866_general_ci](#) (default)
- [koi8r](#) (KOI8-R Relcom Russian) collations:
 - [koi8r_bin](#)
 - [koi8r_general_ci](#) (default)

- `koi8u` (KOI8-U Ukrainian) collations:
 - `koi8u_bin`
 - `koi8u_general_ci` (default)

9.11.7. Asian Character Sets

The Asian character sets that we support include Chinese, Japanese, Korean, and Thai. These can be complicated. For example, the Chinese sets must allow for thousands of different characters. See [Section 9.11.7.1, “The cp932 Character Set”](#), for additional information about the `cp932` and `sjis` character sets.

For answers to some common questions and problems relating support for Asian character sets in MySQL, see [Section A.11, “MySQL 5.1 FAQ — MySQL Chinese, Japanese, and Korean Character Sets”](#).

- `big5` (Big5 Traditional Chinese) collations:
 - `big5_bin`
 - `big5_chinese_ci` (default)
- `cp932` (SJIS for Windows Japanese) collations:
 - `cp932_bin`
 - `cp932_japanese_ci` (default)
- `eucjpms` (UJIS for Windows Japanese) collations:
 - `eucjpms_bin`
 - `eucjpms_japanese_ci` (default)
- `euckr` (EUC-KR Korean) collations:
 - `euckr_bin`
 - `euckr_korean_ci` (default)
- `gb2312` (GB2312 Simplified Chinese) collations:
 - `gb2312_bin`
 - `gb2312_chinese_ci` (default)
- `gbk` (GBK Simplified Chinese) collations:
 - `gbk_bin`
 - `gbk_chinese_ci` (default)
- `sjis` (Shift-JIS Japanese) collations:
 - `sjis_bin`
 - `sjis_japanese_ci` (default)
- `tis620` (TIS620 Thai) collations:
 - `tis620_bin`
 - `tis620_thai_ci` (default)
- `ujis` (EUC-JP Japanese) collations:

- `ujis_bin`
- `ujis_japanese_ci` (default)

9.11.7.1. The `cp932` Character Set

Why is `cp932` needed?

In MySQL, the `sjis` character set corresponds to the `Shift_JIS` character set defined by IANA, which supports JIS X0201 and JIS X0208 characters. (See <http://www.iana.org/assignments/character-sets>.)

However, the meaning of “SHIFT JIS” as a descriptive term has become very vague and it often includes the extensions to `Shift_JIS` that are defined by various vendors.

For example, “SHIFT JIS” used in Japanese Windows environments is a Microsoft extension of `Shift_JIS` and its exact name is `Microsoft Windows Codepage : 932` or `cp932`. In addition to the characters supported by `Shift_JIS`, `cp932` supports extension characters such as NEC special characters, NEC selected — IBM extended characters, and IBM extended characters.

Many Japanese users have experienced problems using these extension characters. These problems stem from the following factors:

- MySQL automatically converts character sets.
- Character sets are converted via Unicode (`ucs2`).
- The `sjis` character set does not support the conversion of these extension characters.
- There are several conversion rules from so-called “SHIFT JIS” to Unicode, and some characters are converted to Unicode differently depending on the conversion rule. MySQL supports only one of these rules (described later).

The MySQL `cp932` character set is designed to solve these problems.

Because MySQL supports character set conversion, it is important to separate IANA `Shift_JIS` and `cp932` into two different character sets because they provide different conversion rules.

How does `cp932` differ from `sjis`?

The `cp932` character set differs from `sjis` in the following ways:

- `cp932` supports NEC special characters, NEC selected — IBM extended characters, and IBM selected characters.
- Some `cp932` characters have two different code points, both of which convert to the same Unicode code point. When converting from Unicode back to `cp932`, one of the code points must be selected. For this “round trip conversion,” the rule recommended by Microsoft is used. (See <http://support.microsoft.com/kb/170559/EN-US/>.)

The conversion rule works like this:

- If the character is in both JIS X 0208 and NEC special characters, use the code point of JIS X 0208.
- If the character is in both NEC special characters and IBM selected characters, use the code point of NEC special characters.
- If the character is in both IBM selected characters and NEC selected — IBM extended characters, use the code point of IBM extended characters.

The table shown at <http://www.microsoft.com/globaldev/reference/dbcs/932.htm> provides information about the Unicode values of `cp932` characters. For `cp932` table entries with characters under which a four-digit number appears, the number represents the corresponding Unicode (`ucs2`) encoding. For table entries with an underlined two-digit value appears, there is a range of `cp932` character values that begin with those two digits. Clicking such a table entry takes you to a page that displays the Unicode value for each of the `cp932` characters that begin with those digits.

The following links are of special interest. They correspond to the encodings for the following sets of characters:

- NEC special characters:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_87.htm

- NEC selected — IBM extended characters:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_ED.htm
http://www.microsoft.com/globaldev/reference/dbcs/932/932_EE.htm

- IBM selected characters:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_FA.htm
http://www.microsoft.com/globaldev/reference/dbcs/932/932_FB.htm
http://www.microsoft.com/globaldev/reference/dbcs/932/932_FC.htm

- **cp932** supports conversion of user-defined characters in combination with **eucjpm**s, and solves the problems with **sjis**/**ujis** conversion. For details, please refer to <http://www.opengroup.or.jp/jvc/cde/sjis-euc-e.html>.

For some characters, conversion to and from **ucs2** is different for **sjis** and **cp932**. The following tables illustrate these differences.

Conversion to **ucs2**:

sjis/cp932 Value	sjis -> ucs2 Conversion	cp932 -> ucs2 Conversion
5C	005C	005C
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225
817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1
81CA	00AC	FFE2

Conversion from **ucs2**:

ucs2 value	ucs2 -> sjis Conversion	ucs2 -> cp932 Conversion
005C	815F	5C
007E	7E	7E
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191

FFE1	3F	8192
FFE2	3F	81CA

Users of any Japanese character sets should be aware that using `--character-set-client-handshake` (or `--skip-character-set-client-handshake`) has an important effect. See [Section 5.2.2, “Command Options”](#).

Chapter 10. Data Types

MySQL supports a number of data types in several categories: numeric types, date and time types, and string (character) types. This chapter first gives an overview of these data types, and then provides a more detailed description of the properties of the types in each category, and a summary of the data type storage requirements. The initial overview is intentionally brief. The more detailed descriptions later in the chapter should be consulted for additional information about particular data types, such as the allowable formats in which you can specify values.

MySQL also supports extensions for handling spatial data. [Chapter 18, *Spatial Extensions*](#), provides information about these data types.

Several of the data type descriptions use these conventions:

- *M* indicates the maximum display width for integer types. For floating-point and fixed-point types, *M* is the total number of digits that can be stored. For string types, *M* is the maximum length. The maximum allowable value of *M* depends on the data type.
- *D* applies to floating-point and fixed-point types and indicates the number of digits following the decimal point. The maximum possible value is 30, but should be no greater than *M*–2.
- Square brackets (‘[’ and ‘]’) indicate optional parts of type definitions.

10.1. Data Type Overview

10.1.1. Overview of Numeric Types

A summary of the numeric data types follows. For additional information, see [Section 10.2, “Numeric Types”](#). Storage requirements are given in [Section 10.5, “Data Type Storage Requirements”](#).

M indicates the maximum display width for integer types. The maximum legal display width is 255. Display width is unrelated to the range of values a type can contain, as described in [Section 10.2, “Numeric Types”](#). For floating-point and fixed-point types, *M* is the total number of digits that can be stored.

If you specify **ZEROFILL** for a numeric column, MySQL automatically adds the **UNSIGNED** attribute to the column.

Numeric data types that allow the **UNSIGNED** attribute also allow **SIGNED**. However, these data types are signed by default, so the **SIGNED** attribute has no effect.

SERIAL is an alias for **BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE**.

SERIAL DEFAULT VALUE in the definition of an integer column is an alias for **NOT NULL AUTO_INCREMENT UNIQUE**.

Warning

When you use subtraction between integer values where one is of type **UNSIGNED**, the result is unsigned unless the **NO_UNSIGNED_SUBTRACTION** SQL mode is enabled. See [Section 11.9, “Cast Functions and Operators”](#).

- **BIT[(M)]**

A bit-field type. *M* indicates the number of bits per value, from 1 to 64. The default is 1 if *M* is omitted.

- **TINYINT[(M)] [UNSIGNED] [ZEROFILL]**

A very small integer. The signed range is **-128** to **127**. The unsigned range is **0** to **255**.

- **BOOL, BOOLEAN**

These types are synonyms for **TINYINT(1)**. A value of zero is considered false. Non-zero values are considered true:

```
mysql> SELECT IF(0, 'true', 'false');
+-----+
| IF(0, 'true', 'false') |
+-----+
| false                  |
+-----+
```

```
mysql> SELECT IF(1, 'true', 'false');
+-----+
| IF(1, 'true', 'false') |
+-----+
| true                   |
+-----+

mysql> SELECT IF(2, 'true', 'false');
+-----+
| IF(2, 'true', 'false') |
+-----+
| true                   |
+-----+
```

However, the values `TRUE` and `FALSE` are merely aliases for `1` and `0`, respectively, as shown here:

```
mysql> SELECT IF(0 = FALSE, 'true', 'false');
+-----+
| IF(0 = FALSE, 'true', 'false') |
+-----+
| true                           |
+-----+

mysql> SELECT IF(1 = TRUE, 'true', 'false');
+-----+
| IF(1 = TRUE, 'true', 'false') |
+-----+
| true                           |
+-----+

mysql> SELECT IF(2 = TRUE, 'true', 'false');
+-----+
| IF(2 = TRUE, 'true', 'false') |
+-----+
| false                          |
+-----+

mysql> SELECT IF(2 = FALSE, 'true', 'false');
+-----+
| IF(2 = FALSE, 'true', 'false') |
+-----+
| false                          |
+-----+
```

The last two statements display the results shown because `2` is equal to neither `1` nor `0`.

We intend to implement full boolean type handling, in accordance with standard SQL, in a future MySQL release.

- **`SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`**

A small integer. The signed range is `-32768` to `32767`. The unsigned range is `0` to `65535`.

- **`MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`**

A medium-sized integer. The signed range is `-8388608` to `8388607`. The unsigned range is `0` to `16777215`.

- **`INT[(M)] [UNSIGNED] [ZEROFILL]`**

A normal-size integer. The signed range is `-2147483648` to `2147483647`. The unsigned range is `0` to `4294967295`.

- **`INTEGER[(M)] [UNSIGNED] [ZEROFILL]`**

This type is a synonym for `INT`.

- **`BIGINT[(M)] [UNSIGNED] [ZEROFILL]`**

A large integer. The signed range is `-9223372036854775808` to `9223372036854775807`. The unsigned range is `0` to `18446744073709551615`.

`SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

Some things you should be aware of with respect to `BIGINT` columns:

- All arithmetic is done using signed `BIGINT` or `DOUBLE` values, so you should not use unsigned big integers larger than

9223372036854775807 (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a **BIGINT** value to a **DOUBLE**.

MySQL can handle **BIGINT** in the following cases:

- When using integers to store large unsigned values in a **BIGINT** column.
- In **MIN(*col_name*)** or **MAX(*col_name*)**, where *col_name* refers to a **BIGINT** column.
- When using operators (+, -, *, and so on) where both operands are integers.
- You can always store an exact integer value in a **BIGINT** column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.
- The -, +, and * operators use **BIGINT** arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than 9223372036854775807.
- **FLOAT[(*M*,*D*)] [UNSIGNED] [ZEROFILL]**

A small (single-precision) floating-point number. Allowable values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

M is the total number of digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits allowed by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

UNSIGNED, if specified, disallows negative values.

Using **FLOAT** might give you some unexpected problems because all calculations in MySQL are done with double precision. See [Section B.1.5.7, “Solving Problems with No Matching Rows”](#).

- **DOUBLE[(*M*,*D*)] [UNSIGNED] [ZEROFILL]**
A normal-size (double-precision) floating-point number. Allowable values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.
M is the total number of digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits allowed by the hardware. A double-precision floating-point number is accurate to approximately 15 decimal places.
UNSIGNED, if specified, disallows negative values.

- **DOUBLE PRECISION[(*M*,*D*)] [UNSIGNED] [ZEROFILL], **REAL[(*M*,*D*)] [UNSIGNED] [ZEROFILL]**
These types are synonyms for **DOUBLE**. Exception: If the **REAL_AS_FLOAT** SQL mode is enabled, **REAL** is a synonym for **FLOAT** rather than **DOUBLE**.**
- **FLOAT(*p*) [UNSIGNED] [ZEROFILL]**

A floating-point number. *p* represents the precision in bits, but MySQL uses this value only to determine whether to use **FLOAT** or **DOUBLE** for the resulting data type. If *p* is from 0 to 24, the data type becomes **FLOAT** with no *M* or *D* values. If *p* is from 25 to 53, the data type becomes **DOUBLE** with no *M* or *D* values. The range of the resulting column is the same as for the single-precision **FLOAT** or double-precision **DOUBLE** data types described earlier in this section.

FLOAT(*p*) syntax is provided for ODBC compatibility.

- **DECIMAL[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]**
A packed “exact” fixed-point number. *M* is the total number of digits (the precision) and *D* is the number of digits after the decimal point (the scale). The decimal point and (for negative numbers) the ‘-’ sign are not counted in *M*. If *D* is 0, values have no decimal point or fractional part. The maximum number of digits (*M*) for **DECIMAL** is 65. The maximum number of supported decimals (*D*) is 30. If *D* is omitted, the default is 0. If *M* is omitted, the default is 10.

UNSIGNED, if specified, disallows negative values.

All basic calculations (+, -, *, /) with **DECIMAL** columns are done with a precision of 65 digits.

- **DEC[(M[,D])] [UNSIGNED] [ZEROFILL]**, **NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]**, **FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]**

These types are synonyms for **DECIMAL**. The **FIXED** synonym is available for compatibility with other database systems.

10.1.2. Overview of Date and Time Types

A summary of the temporal data types follows. For additional information, see [Section 10.3, “Date and Time Types”](#). Storage requirements are given in [Section 10.5, “Data Type Storage Requirements”](#). Functions that operate on temporal values are described at [Section 11.6, “Date and Time Functions”](#).

For the **DATETIME** and **DATE** range descriptions, “supported” means that although earlier values might work, there is no guarantee.

- **DATE**

A date. The supported range is '1000-01-01' to '9999-12-31'. MySQL displays **DATE** values in 'YYYY-MM-DD' format, but allows assignment of values to **DATE** columns using either strings or numbers.

- **DATETIME**

A date and time combination. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. MySQL displays **DATETIME** values in 'YYYY-MM-DD HH:MM:SS' format, but allows assignment of values to **DATETIME** columns using either strings or numbers.

- **TIMESTAMP**

A timestamp. The range is '1970-01-01 00:00:01' UTC to partway through the year 2038. **TIMESTAMP** values are stored as the number of seconds since the epoch ('1970-01-01 00:00:00' UTC). A **TIMESTAMP** cannot represent the value '1970-01-01 00:00:00' because that is equivalent to 0 seconds from the epoch and the value 0 is reserved for representing '0000-00-00 00:00:00', the “zero” **TIMESTAMP** value.

A **TIMESTAMP** column is useful for recording the date and time of an **INSERT** or **UPDATE** operation. By default, the first **TIMESTAMP** column in a table is automatically set to the date and time of the most recent operation if you do not assign it a value yourself. You can also set any **TIMESTAMP** column to the current date and time by assigning it a **NULL** value. Variations on automatic initialization and update properties are described in [Section 10.3.1.1, “TIMESTAMP Properties”](#).

A **TIMESTAMP** value is returned as a string in the format 'YYYY-MM-DD HH:MM:SS' with a display width fixed at 19 characters. To obtain the value as a number, you should add +0 to the timestamp column.

Note

The **TIMESTAMP** format that was used prior to MySQL 4.1 is not supported in MySQL 5.1; see *MySQL 3.23, 4.0, 4.1 Reference Manual* for information regarding the old format.

- **TIME**

A time. The range is '-838:59:59' to '838:59:59'. MySQL displays **TIME** values in 'HH:MM:SS' format, but allows assignment of values to **TIME** columns using either strings or numbers.

- **YEAR[(2|4)]**

A year in two-digit or four-digit format. The default is four-digit format. In four-digit format, the allowable values are 1901 to 2155, and 0000. In two-digit format, the allowable values are 70 to 69, representing years from 1970 to 2069. MySQL displays **YEAR** values in YYYY format, but allows you to assign values to **YEAR** columns using either strings or numbers.

The **SUM()** and **AVG()** aggregate functions do not work with temporal values. (They convert the values to numbers, which loses the part after the first non-numeric character.) To work around this problem, you can convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

10.1.3. Overview of String Types

A summary of the string data types follows. For additional information, see [Section 10.4, “String Types”](#). Storage requirements are given in [Section 10.5, “Data Type Storage Requirements”](#).

In some cases, MySQL may change a string column to a type different from that given in a `CREATE TABLE` or `ALTER TABLE` statement. See [Section 12.1.10.1, “Silent Column Specification Changes”](#).

MySQL interprets length specifications in character column definitions in character units. This applies to `CHAR`, `VARCHAR`, and the `TEXT` types.

Column definitions for many string data types can include attributes that specify the character set or collation of the column. These attributes apply to the `CHAR`, `VARCHAR`, the `TEXT` types, `ENUM`, and `SET` data types:

- The `CHARACTER SET` attribute specifies the character set, and the `COLLATE` attribute specifies a collation for the character set. For example:

```
CREATE TABLE t
(
  c1 VARCHAR(20) CHARACTER SET utf8,
  c2 TEXT CHARACTER SET latin1 COLLATE latin1_general_cs
);
```

This table definition creates a column named `c1` that has a character set of `utf8` with the default collation for that character set, and a column named `c2` that has a character set of `latin1` and a case-sensitive collation.

`CHARSET` is a synonym for `CHARACTER SET`.

- The `ASCII` attribute is shorthand for `CHARACTER SET latin1`.
- The `UNICODE` attribute is shorthand for `CHARACTER SET ucs2`.
- The `BINARY` attribute is shorthand for specifying the binary collation of the column character set. In this case, sorting and comparison are based on numeric character values.

Character column sorting and comparison are based on the character set assigned to the column. For the `CHAR`, `VARCHAR`, `TEXT`, `ENUM`, and `SET` data types, you can declare a column with a binary collation or the `BINARY` attribute to cause sorting and comparison to use the underlying character code values rather than a lexical ordering.

[Chapter 9, Character Set Support](#), provides additional information about use of character sets in MySQL.

- `[NATIONAL] CHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

A fixed-length string that is always right-padded with spaces to the specified length when stored. *M* represents the column length. The range of *M* is 0 to 255 characters.

Note

Trailing spaces are removed when `CHAR` values are retrieved.

If you attempt to set the length of a `CHAR` greater than 255, the `CREATE TABLE` or `ALTER TABLE` statement in which this is done fails with an error:

```
mysql> CREATE TABLE c1 (col1 INT, col2 CHAR(500));
ERROR 1074 (42000): Column length too big for column 'col' (max = 255);
use BLOB or TEXT instead
mysql> SHOW CREATE TABLE c1;
ERROR 1146 (42S02): Table 'test.c1' doesn't exist
```

`CHAR` is shorthand for `CHARACTER`. `NATIONAL CHAR` (or its equivalent short form, `NCHAR`) is the standard SQL way to define that a `CHAR` column should use some predefined character set. MySQL 4.1 and up uses `utf8` as this predefined character set. [Sec-](#)

tion 9.3.6, “National Character Set”.

The **CHAR BYTE** data type is an alias for the **BINARY** data type. This is a compatibility feature.

MySQL allows you to create a column of type **CHAR(0)**. This is useful primarily when you have to be compliant with old applications that depend on the existence of a column but that do not actually use its value. **CHAR(0)** is also quite nice when you need a column that can take only two values: A column that is defined as **CHAR(0) NULL** occupies only one bit and can take only the values **NULL** and **''** (the empty string).

- **CHAR [CHARACTER SET *charset_name*] [COLLATE *collation_name*]**

This type is a synonym for **CHAR(1)**.

- **[NATIONAL] VARCHAR(*M*) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]**

A variable-length string. *M* represents the maximum column length. The range of *M* is 0 to 65,535. (The actual maximum length of a **VARCHAR** is determined by the maximum row size and the character set you use. The maximum *effective* column length is subject to a row size of 65,532 bytes.)

Note

MySQL 5.1 follows the standard SQL specification, and does *not* remove trailing spaces from **VARCHAR** values.

VARCHAR is shorthand for **CHARACTER VARYING**.

VARCHAR is stored with a one-byte or two-byte length prefix plus data. The length prefix is two bytes if the **VARCHAR** column is declared with a length greater than 255.

- **BINARY(*M*)**

The **BINARY** type is similar to the **CHAR** type, but stores binary byte strings rather than non-binary character strings.

- **VARBINARY(*M*)**

The **VARBINARY** type is similar to the **VARCHAR** type, but stores binary byte strings rather than non-binary character strings.

- **TINYBLOB**

A **BLOB** column with a maximum length of 255 ($2^8 - 1$) bytes.

- **TINYTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]**

A **TEXT** column with a maximum length of 255 ($2^8 - 1$) characters.

- **BLOB[(*M*)]**

A **BLOB** column with a maximum length of 65,535 ($2^{16} - 1$) bytes.

An optional length *M* can be given for this type. If this is done, MySQL creates the column as the smallest **BLOB** type large enough to hold values *M* bytes long.

- **TEXT[(*M*)] [CHARACTER SET *charset_name*] [COLLATE *collation_name*]**

A **TEXT** column with a maximum length of 65,535 ($2^{16} - 1$) characters.

An optional length *M* can be given for this type. If this is done, MySQL creates the column as the smallest **TEXT** type large enough to hold values *M* characters long.

- **MEDIUMBLOB**

A **BLOB** column with a maximum length of 16,777,215 ($2^{24} - 1$) bytes.

- **MEDIUMTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]**

A **TEXT** column with a maximum length of 16,777,215 ($2^{24} - 1$) characters.

- **LONGBLOB**

A **BLOB** column with a maximum length of 4,294,967,295 or 4GB ($2^{32} - 1$) bytes. The maximum *effective* (permitted) length of **LONGBLOB** columns depends on the configured maximum packet size in the client/server protocol and available memory.

- **LONGTEXT** [**CHARACTER SET** *charset_name*] [**COLLATE** *collation_name*]

A **TEXT** column with a maximum length of 4,294,967,295 or 4GB ($2^{32} - 1$) characters. The maximum *effective* (permitted) length of **LONGTEXT** columns depends on the configured maximum packet size in the client/server protocol and available memory.

- **ENUM**('value1', 'value2', ...) [**CHARACTER SET** *charset_name*] [**COLLATE** *collation_name*]

An enumeration. A string object that can have only one value, chosen from the list of values 'value1', 'value2', ..., **NULL** or the special '' error value. An **ENUM** column can have a maximum of 65,535 distinct values. **ENUM** values are represented internally as integers.

- **SET**('value1', 'value2', ...) [**CHARACTER SET** *charset_name*] [**COLLATE** *collation_name*]

A set. A string object that can have zero or more values, each of which must be chosen from the list of values 'value1', 'value2', ... A **SET** column can have a maximum of 64 members. **SET** values are represented internally as integers.

10.1.4. Data Type Default Values

The **DEFAULT** *value* clause in a data type specification indicates a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as **NOW()** or **CURRENT_DATE**. The exception is that you can specify **CURRENT_TIMESTAMP** as the default for a **TIMESTAMP** column. See [Section 10.3.1.1, “TIMESTAMP Properties”](#).

BLOB and **TEXT** columns cannot be assigned a default value.

If a column definition includes no explicit **DEFAULT** value, MySQL determines the default value as follows:

If the column can take **NULL** as a value, the column is defined with an explicit **DEFAULT NULL** clause.

If the column cannot take **NULL** as the value, MySQL defines the column with no explicit **DEFAULT** clause. For data entry, if an **INSERT** or **REPLACE** statement includes no value for the column, MySQL handles the column according to the SQL mode in effect at the time:

- If strict SQL mode is not enabled, MySQL sets the column to the implicit default value for the column data type.
- If strict mode is enabled, an error occurs for transactional tables and the statement is rolled back. For non-transactional tables, an error occurs, but if this happens for the second or subsequent row of a multiple-row statement, the preceding rows will have been inserted.

Suppose that a table **t** is defined as follows:

```
CREATE TABLE t (i INT NOT NULL);
```

In this case, **i** has no explicit default, so in strict mode each of the following statements produce an error and no row is inserted. When not using strict mode, only the third statement produces an error; the implicit default is inserted for the first two statements, but the third fails because **DEFAULT(i)** cannot produce a value:

```
INSERT INTO t VALUES();  
INSERT INTO t VALUES(DEFAULT);  
INSERT INTO t VALUES(DEFAULT(i));
```

See [Section 5.2.6, “SQL Modes”](#).

For a given table, you can use the **SHOW CREATE TABLE** statement to see which columns have an explicit **DEFAULT** clause.

Implicit defaults are defined as follows:

- For numeric types, the default is `0`, with the exception that for integer types declared with the `AUTO_INCREMENT` attribute, the default is the next value in the sequence.
- For date and time types other than `TIMESTAMP`, the default is the appropriate “zero” value for the type. For the first `TIMESTAMP` column in a table, the default value is the current date and time. See [Section 10.3, “Date and Time Types”](#).
- For string types other than `ENUM`, the default value is the empty string. For `ENUM`, the default is the first enumeration value.

10.2. Numeric Types

MySQL supports all of the standard SQL numeric data types. These types include the exact numeric data types (`INTEGER`, `SMALLINT`, `DECIMAL`, and `NUMERIC`), as well as the approximate numeric data types (`FLOAT`, `REAL`, and `DOUBLE PRECISION`). The keyword `INT` is a synonym for `INTEGER`, and the keyword `DEC` is a synonym for `DECIMAL`. For numeric type storage requirements, see [Section 10.5, “Data Type Storage Requirements”](#).

The `BIT` data type stores bit-field values and is supported for `MyISAM`, `MEMORY`, and `InnoDB` tables.

As an extension to the SQL standard, MySQL also supports the integer types `TINYINT`, `MEDIUMINT`, and `BIGINT`. The following table shows the required storage and range for each of the integer types.

Type	Bytes	Minimum Value (Signed/Unsigned)	Maximum Value (Signed/Unsigned)
<code>TINYINT</code>	1	-128 0	127 255
<code>SMALLINT</code>	2	-32768 0	32767 65535
<code>MEDIUMINT</code>	3	-8388608 0	8388607 16777215
<code>INT</code>	4	-2147483648 0	2147483647 4294967295
<code>BIGINT</code>	8	-9223372036854775808 0	9223372036854775807 18446744073709551615

Another extension is supported by MySQL for optionally specifying the display width of integer data types in parentheses following the base keyword for the type (for example, `INT(4)`). This optional display width is used to display integer values having a width less than the width specified for the column by left-padding them with spaces.

The display width does *not* constrain the range of values that can be stored in the column, nor the number of digits that are displayed for values having a width exceeding that specified for the column. For example, a column specified as `SMALLINT(3)` has the usual `SMALLINT` range of -32768 to 32767, and values outside the range allowed by three characters are displayed using more than three characters.

When used in conjunction with the optional extension attribute `ZEROFILL`, the default padding of spaces is replaced with zeros. For example, for a column declared as `INT(5) ZEROFILL`, a value of 4 is retrieved as 00004. Note that if you store larger values than the display width in an integer column, you may experience problems when MySQL generates temporary tables for some complicated joins, because in these cases MySQL assumes that the data fits into the original column width.

Note

The `ZEROFILL` attribute is stripped when a column is involved in expressions or `UNION` queries.

All integer types can have an optional (non-standard) attribute `UNSIGNED`. Unsigned values can be used when you want to allow only non-negative numbers in a column and you need a larger upper numeric range for the column. For example, if an `INT` column is `UNSIGNED`, the size of the column's range is the same but its endpoints shift from -2147483648 and 2147483647 up to 0 and 4294967295.

Floating-point and fixed-point types also can be `UNSIGNED`. As with integer types, this attribute prevents negative values from being

stored in the column. However, unlike the integer types, the upper range of column values remains the same.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

Integer data types can have the additional attribute `AUTO_INCREMENT`. When you insert a value of `NULL` (recommended) or `0` into an indexed `AUTO_INCREMENT` column, the column is set to the next sequence value. Typically this is `value+1`, where `value` is the largest value for the column currently in the table. `AUTO_INCREMENT` sequences begin with `1`.

For floating-point data types, MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

The `FLOAT` and `DOUBLE` data types are used to represent approximate numeric data values. For `FLOAT`, the SQL standard allows an optional specification of the precision (but not the range of the exponent) in bits following the keyword `FLOAT` in parentheses. MySQL also supports this optional precision specification, but the precision value is used only to determine storage size. A precision from 0 to 23 results in a four-byte single-precision `FLOAT` column. A precision from 24 to 53 results in an eight-byte double-precision `DOUBLE` column.

MySQL allows a non-standard syntax: `FLOAT(M,D)` or `REAL(M,D)` or `DOUBLE PRECISION(M,D)`. Here, “(M,D)” means that values can be stored with up to `M` digits in total, of which `D` digits may be after the decimal point. For example, a column defined as `FLOAT(7,4)` will look like `-999.9999` when displayed. MySQL performs rounding when storing values, so if you insert `999.00009` into a `FLOAT(7,4)` column, the approximate result is `999.0001`.

MySQL treats `DOUBLE` as a synonym for `DOUBLE PRECISION` (a non-standard extension). MySQL also treats `REAL` as a synonym for `DOUBLE PRECISION` (a non-standard variation), unless the `REAL_AS_FLOAT` SQL mode is enabled.

For maximum portability, code requiring storage of approximate numeric data values should use `FLOAT` or `DOUBLE PRECISION` with no specification of precision or number of digits.

The `DECIMAL` and `NUMERIC` data types are used to store exact numeric data values. In MySQL, `NUMERIC` is implemented as `DECIMAL`. These types are used to store values for which it is important to preserve exact precision, for example with monetary data.

MySQL 5.1 stores `DECIMAL` and `NUMERIC` values in binary format. Before MySQL 5.0.3, they were stored as strings. See [Chapter 24, Precision Math](#).

When declaring a `DECIMAL` or `NUMERIC` column, the precision and scale can be (and usually is) specified; for example:

```
salary DECIMAL(5,2)
```

In this example, `5` is the precision and `2` is the scale. The precision represents the number of significant digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point. If the scale is 0, `DECIMAL` and `NUMERIC` values contain no decimal point or fractional part.

Standard SQL requires that the `salary` column be able to store any value with five digits and two decimals. In this case, therefore, the range of values that can be stored in the `salary` column is from `-999.99` to `999.99`.

In standard SQL, the syntax `DECIMAL(M)` is equivalent to `DECIMAL(M,0)`. Similarly, the syntax `DECIMAL` is equivalent to `DECIMAL(M,0)`, where the implementation is allowed to decide the value of `M`. MySQL supports both of these variant forms of the `DECIMAL` and `NUMERIC` syntax. The default value of `M` is 10.

The maximum number of digits for `DECIMAL` or `NUMERIC` is 65, but the actual range for a given `DECIMAL` or `NUMERIC` column can be constrained by the precision or scale for a given column. When such a column is assigned a value with more digits following the decimal point than are allowed by the specified scale, the value is converted to that scale. (The precise behavior is operating system-specific, but generally the effect is truncation to the allowable number of digits.)

The `BIT` data type is used to store bit-field values. A type of `BIT(M)` allows for storage of `M`-bit values. `M` can range from 1 to 64.

To specify bit values, `b'value'` notation can be used. `value` is a binary value written using zeros and ones. For example, `b'111'` and `b'10000000'` represent 7 and 128, respectively. See [Section 8.1.5, “Bit-Field Values”](#).

If you assign a value to a `BIT(M)` column that is less than `M` bits long, the value is padded on the left with zeros. For example, assigning a value of `b'101'` to a `BIT(6)` column is, in effect, the same as assigning `b'000101'`.

When asked to store a value in a numeric column that is outside the data type's allowable range, MySQL's behavior depends on the SQL mode in effect at the time. For example, if no restrictive modes are enabled, MySQL clips the value to the appropriate endpoint of the range and stores the resulting value instead. However, if the mode is set to `TRADITIONAL`, MySQL rejects a value that is out of range with an error, and the insert fails, in accordance with the SQL standard.

In non-strict mode, when an out-of-range value is assigned to an integer column, MySQL stores the value representing the corresponding endpoint of the column data type range. If you store 256 into a `TINYINT` or `TINYINT UNSIGNED` column, MySQL stores 127 or 255, respectively. When a floating-point or fixed-point column is assigned a value that exceeds the range implied by the specified (or default) precision and scale, MySQL stores the value representing the corresponding endpoint of that range.

Conversions that occur due to clipping when MySQL is not operating in strict mode are reported as warnings for `ALTER TABLE`, `LOAD DATA INFILE`, `UPDATE`, and multiple-row `INSERT` statements. When MySQL is operating in strict mode, these statements fail, and some or all of the values will not be inserted or changed, depending on whether the table is a transactional table and other factors. For details, see [Section 5.2.6, “SQL Modes”](#).

10.3. Date and Time Types

The date and time types for representing temporal values are `DATETIME`, `DATE`, `TIMESTAMP`, `TIME`, and `YEAR`. Each temporal type has a range of legal values, as well as a “zero” value that may be used when you specify an illegal value that MySQL cannot represent. The `TIMESTAMP` type has special automatic updating behavior, described later on. For temporal type storage requirements, see [Section 10.5, “Data Type Storage Requirements”](#).

MySQL gives warnings or errors if you try to insert an illegal date. By setting the SQL mode to the appropriate value, you can specify more exactly what kind of dates you want MySQL to support. (See [Section 5.2.6, “SQL Modes”](#).) You can get MySQL to accept certain dates, such as `'1999-11-31'`, by using the `ALLOW_INVALID_DATES` SQL mode. This is useful when you want to store a “possibly wrong” value which the user has specified (for example, in a web form) in the database for future processing. Under this mode, MySQL verifies only that the month is in the range from 0 to 12 and that the day is in the range from 0 to 31. These ranges are defined to include zero because MySQL allows you to store dates where the day or month and day are zero in a `DATE` or `DATETIME` column. This is extremely useful for applications that need to store a birthdate for which you do not know the exact date. In this case, you simply store the date as `'1999-00-00'` or `'1999-01-00'`. If you store dates such as these, you should not expect to get correct results for functions such as `DATE_SUB()` or `DATE_ADD` that require complete dates. (If you do *not* want to allow zero in dates, you can use the `NO_ZERO_IN_DATE` SQL mode).

Prior to MySQL 5.1.18, when `DATE` values are compared with `DATETIME` values the time portion of the `DATETIME` value is ignored. Starting from MySQL 5.1.18, a `DATE` value is coerced to the `DATETIME` type by adding the time portion as `'00:00:00'`. To mimic the old behavior, use the `CAST()` function to perform the comparison in the following way:

```
date_col = CAST(NOW() as DATE);
```

MySQL also allows you to store `'0000-00-00'` as a “dummy date” (if you are not using the `NO_ZERO_DATE` SQL mode). This is in some cases more convenient (and uses less data and index space) than using `NULL` values.

Here are some general considerations to keep in mind when working with date and time types:

- MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to be assigned to or compared to a date or time type). Only the formats described in the following sections are supported. It is expected that you supply legal values. Unpredictable results may occur if you use values in other formats.
- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using the following rules:
 - Year values in the range `70-99` are converted to `1970-1999`.
 - Year values in the range `00-69` are converted to `2000-2069`.
- Although MySQL tries to interpret values in several formats, dates always must be given in year-month-day order (for example, `'98-09-04'`), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, `'09-04-98'`, `'04-09-98'`).
- MySQL automatically converts a date or time type value to a number if the value is used in a numeric context and vice versa.
- By default, when MySQL encounters a value for a date or time type that is out of range or otherwise illegal for the type (as described at the beginning of this section), it converts the value to the “zero” value for that type. The exception is that out-of-range `TIME` values are clipped to the appropriate endpoint of the `TIME` range.

The following table shows the format of the “zero” value for each type. Note that the use of these values produces warnings if the `NO_ZERO_DATE` SQL mode is enabled.

Data Type	“Zero” Value
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000

- The “zero” values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values '0' or 0, which are easier to write.
- “Zero” date or time values used through MyODBC are converted automatically to **NULL** in MyODBC 2.50.12 and above, because ODBC cannot handle such values.

10.3.1. The DATETIME, DATE, and TIMESTAMP Types

The **DATETIME**, **DATE**, and **TIMESTAMP** types are related. This section describes their characteristics, how they are similar, and how they differ.

The **DATETIME** type is used when you need values that contain both date and time information. MySQL retrieves and displays **DATE-TIME** values in 'YYYY-MM-DD HH:MM:SS' format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

The **DATE** type is used when you need only a date value, without a time part. MySQL retrieves and displays **DATE** values in 'YYYY-MM-DD' format. The supported range is '1000-01-01' to '9999-12-31'.

For the **DATETIME** and **DATE** range descriptions, “supported” means that although earlier values might work, there is no guarantee.

The **TIMESTAMP** data type has varying properties, depending on the MySQL version and the SQL mode the server is running in. These properties are described later in this section.

You can specify **DATETIME**, **DATE**, and **TIMESTAMP** values using any of a common set of formats:

- As a string in either 'YYYY-MM-DD HH:MM:SS' or 'YY-MM-DD HH:MM:SS' format. A “relaxed” syntax is allowed: Any punctuation character may be used as the delimiter between date parts or time parts. For example, '98-12-31 11:30:45', '98.12.31 11+30+45', '98/12/31 11*30*45', and '98@12@31 11^30^45' are equivalent.
- As a string in either 'YYYY-MM-DD' or 'YY-MM-DD' format. A “relaxed” syntax is allowed here, too. For example, '98-12-31', '98.12.31', '98/12/31', and '98@12@31' are equivalent.
- As a string with no delimiters in either 'YYYYMMDDHHMMSS' or 'YYMMDDHHMMSS' format, provided that the string makes sense as a date. For example, '19970523091528' and '970523091528' are interpreted as '1997-05-23 09:15:28', but '971122129015' is illegal (it has a nonsensical minute part) and becomes '0000-00-00 00:00:00'.
- As a string with no delimiters in either 'YYYYMMDD' or 'YYMMDD' format, provided that the string makes sense as a date. For example, '19970523' and '970523' are interpreted as '1997-05-23', but '971332' is illegal (it has nonsensical month and day parts) and becomes '0000-00-00'.
- As a number in either YYYYMMDDHHMMSS or YYMMDDHHMMSS format, provided that the number makes sense as a date. For example, 19830905132800 and 830905132800 are interpreted as '1983-09-05 13:28:00'.
- As a number in either YYYYMMDD or YYMMDD format, provided that the number makes sense as a date. For example, 19830905 and 830905 are interpreted as '1983-09-05'.
- As the result of a function that returns a value that is acceptable in a **DATETIME**, **DATE**, or **TIMESTAMP** context, such as **NOW()** or **CURRENT_DATE**.

A microseconds part is allowable in temporal values in some contexts, such as in literal values, and in the arguments to or return values from some temporal functions. Microseconds are specified as a trailing .uuuuuu part in the value. Example:

```
mysql> SELECT MICROSECOND('2010-12-10 14:12:09.019473');
+-----+
| MICROSECOND('2010-12-10 14:12:09.019473') |
+-----+
| 19473 |
+-----+
```

However, microseconds cannot be stored into a column of any temporal data type. Any microseconds part is discarded.

Conversion of **DATETIME** values to numeric form (for example, by adding **+0**) results in a double value with a microseconds part of **.000000**:

```
mysql> SELECT NOW(), NOW()+0;
+-----+-----+
| NOW() | NOW()+0 |
+-----+-----+
| 2007-04-23 14:21:52 | 20070423142152.000000 |
+-----+-----+
```

Illegal **DATETIME**, **DATE**, or **TIMESTAMP** values are converted to the “zero” value of the appropriate type ('**0000-00-00 00:00:00**' or '**0000-00-00**').

For values specified as strings that include date part delimiters, it is not necessary to specify two digits for month or day values that are less than **10**. '**1979-6-9**' is the same as '**1979-06-09**'. Similarly, for values specified as strings that include time part delimiters, it is not necessary to specify two digits for hour, minute, or second values that are less than **10**. '**1979-10-30 1:2:3**' is the same as '**1979-10-30 01:02:03**'.

Values specified as numbers should be 6, 8, 12, or 14 digits long. If a number is 8 or 14 digits long, it is assumed to be in **YYYYMMDD** or **YYYYMMDDHHMMSS** format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in **YYMMDD** or **YYMMDDHHMMSS** format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as non-delimited strings are interpreted using their length as given. If the string is 8 or 14 characters long, the year is assumed to be given by the first 4 characters. Otherwise, the year is assumed to be given by the first 2 characters. The string is interpreted from left to right to find year, month, day, hour, minute, and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify '**9903**', thinking that represents March, 1999, MySQL inserts a “zero” date value into your table. This occurs because the year and month values are **99** and **03**, but the day part is completely missing, so the value is not a legal date. However, you can explicitly specify a value of zero to represent missing month or day parts. For example, you can use '**990300**' to insert the value '**1999-03-00**'.

You can to some extent assign values of one date type to an object of a different date type. However, there may be some alteration of the value or loss of information:

- If you assign a **DATE** value to a **DATETIME** or **TIMESTAMP** object, the time part of the resulting value is set to '**00:00:00**' because the **DATE** value contains no time information.
- If you assign a **DATETIME** or **TIMESTAMP** value to a **DATE** object, the time part of the resulting value is deleted because the **DATE** type stores no time information.
- Remember that although **DATETIME**, **DATE**, and **TIMESTAMP** values all can be specified using the same set of formats, the types do not all have the same range of values. For example, **TIMESTAMP** values cannot be earlier than **1970** or later than **2038**. This means that a date such as '**1968-01-01**', while legal as a **DATETIME** or **DATE** value, is not valid as a **TIMESTAMP** value and is converted to **0**.

Be aware of certain pitfalls when specifying date values:

- The relaxed format allowed for values specified as strings can be deceiving. For example, a value such as '**10:11:12**' might look like a time value because of the ':' delimiter, but if used in a date context is interpreted as the year '**2010-11-12**'. The value '**10:45:15**' is converted to '**0000-00-00**' because '**45**' is not a legal month.
- The server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as '**2004-04-31**' are converted to '**0000-00-00**' and a warning is generated. With strict mode enabled, invalid dates generate an error. To allow such dates, enable **ALLOW_INVALID_DATES**. See [Section 5.2.6, “SQL Modes”](#), for more information.

- MySQL does not accept timestamp values that include a zero in the day or month column or values that are not a valid date. The sole exception to this rule is the special value `'0000-00-00 00:00:00'`.
- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using the following rules:
 - Year values in the range `00-69` are converted to `2000-2069`.
 - Year values in the range `70-99` are converted to `1970-1999`.

10.3.1.1. **TIMESTAMP** Properties

TIMESTAMP columns are displayed in the same format as **DATETIME** columns. In other words, the display width is fixed at 19 characters, and the format is `'YYYY-MM-DD HH:MM:SS'`.

TIMESTAMP values are converted from the current time zone to UTC for storage, and converted back from UTC to the current time zone for retrieval. (This occurs only for the **TIMESTAMP** data type, not for other types such as **DATETIME**.) By default, the current time zone for each connection is the server's time. The time zone can be set on a per-connection basis, as described in [Section 5.10.8, “MySQL Server Time Zone Support”](#). As long as the time zone setting remains constant, you get back the same value you store. If you store a **TIMESTAMP** value, and then change the time zone and retrieve the value, the retrieved value is different from the value you stored. This occurs because the same time zone was not used for conversion in both directions. The current time zone is available as the value of the `time_zone` system variable.

The **TIMESTAMP** data type offers automatic initialization and updating. You can choose whether to use these properties and which column should have them:

- For one **TIMESTAMP** column in a table, you can assign the current timestamp as the default value and the auto-update value. It is possible to have the current timestamp be the default value for initializing the column, for the auto-update value, or both. It is not possible to have the current timestamp be the default value for one column and the auto-update value for another column.
- Any single **TIMESTAMP** column in a table can be used as the one that is initialized to the current date and time, or updated automatically. This need not be the first **TIMESTAMP** column.
- If a **DEFAULT** value is specified for the first **TIMESTAMP** column in a table, it is not ignored. The default can be **CURRENT_TIMESTAMP** or a constant date and time value.
- In a **CREATE TABLE** statement, the first **TIMESTAMP** column can be declared in any of the following ways:
 - With both **DEFAULT CURRENT_TIMESTAMP** and **ON UPDATE CURRENT_TIMESTAMP** clauses, the column has the current timestamp for its default value, and is automatically updated.
 - With neither **DEFAULT** nor **ON UPDATE** clauses, it is the same as **DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP**.
 - With a **DEFAULT CURRENT_TIMESTAMP** clause and no **ON UPDATE** clause, the column has the current timestamp for its default value but is not automatically updated.
 - With no **DEFAULT** clause and with an **ON UPDATE CURRENT_TIMESTAMP** clause, the column has a default of 0 and is automatically updated.
 - With a constant **DEFAULT** value, the column has the given default and is not automatically initialized to the current timestamp. If the column also has an **ON UPDATE CURRENT_TIMESTAMP** clause, it is automatically updated; otherwise, it has a constant default and is not automatically updated.

In other words, you can use the current timestamp for both the initial value and the auto-update value, or either one, or neither. (For example, you can specify **ON UPDATE** to enable auto-update without also having the column auto-initialized.) The following column definitions demonstrate each of the possibilities:

- Auto-initialization and auto-update:

```
ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
```

- Auto-initialization only:


```
ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

- Auto-update only:

```
ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP
```

- Neither:

```
ts TIMESTAMP DEFAULT 0
```

- To specify automatic default or updating for a **TIMESTAMP** column other than the first one, you must suppress the automatic initialization and update behaviors for the first **TIMESTAMP** column by explicitly assigning it a constant **DEFAULT** value (for example, **DEFAULT 0** or **DEFAULT '2003-01-01 00:00:00'**). Then, for the other **TIMESTAMP** column, the rules are the same as for the first **TIMESTAMP** column, except that if you omit both of the **DEFAULT** and **ON UPDATE** clauses, no automatic initialization or updating occurs.

Example:

```
CREATE TABLE t (
  ts1 TIMESTAMP DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
  ON UPDATE CURRENT_TIMESTAMP);
```

- **CURRENT_TIMESTAMP** or any of its synonyms (**CURRENT_TIMESTAMP()**, **NOW()**, **LOCALTIME**, **LOCALTIME()**, **LOCALTIMESTAMP**, or **LOCALTIMESTAMP()**) can be used in the **DEFAULT** and **ON UPDATE** clauses. They all mean “the current timestamp.” (**UTC_TIMESTAMP** is not allowed. Its range of values does not align with those of the **TIMESTAMP** column anyway unless the current time zone is **UTC**.)
- The order of the **DEFAULT** and **ON UPDATE** attributes does not matter. If both **DEFAULT** and **ON UPDATE** are specified for a **TIMESTAMP** column, either can precede the other. For example, these statements are equivalent:

```
CREATE TABLE t (ts TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
  ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
  DEFAULT CURRENT_TIMESTAMP);
```

TIMESTAMP columns are **NOT NULL** by default, cannot contain **NULL** values, and assigning **NULL** assigns the current timestamp. However, a **TIMESTAMP** column can be allowed to contain **NULL** by declaring it with the **NULL** attribute. In this case, the default value also becomes **NULL** unless overridden with a **DEFAULT** clause that specifies a different default value. **DEFAULT NULL** can be used to explicitly specify **NULL** as the default value. (For a **TIMESTAMP** column not declared with the **NULL** attribute, **DEFAULT NULL** is illegal.) If a **TIMESTAMP** column allows **NULL** values, assigning **NULL** sets it to **NULL**, not to the current timestamp.

The following table contains several **TIMESTAMP** columns that allow **NULL** values:

```
CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
```

Note that a **TIMESTAMP** column that allows **NULL** values will *not* take on the current timestamp except under one of the following conditions:

- Its default value is defined as **CURRENT_TIMESTAMP**
- **NOW()** or **CURRENT_TIMESTAMP** is inserted into the column

In other words, a **TIMESTAMP** column defined as **NULL** will auto-initialize only if it is created using a definition such as the following:

```
CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP);
```


Otherwise — that is, if the `TIMESTAMP` column is defined to allow `NULL` values but not using `DEFAULT CURRENT_TIMESTAMP`, as shown here...

```
CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT NULL);
CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');
```

...then you must explicitly insert a value corresponding to the current date and time. For example:

```
INSERT INTO t1 VALUES (NOW());
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
```

Note

The MySQL server can be run with the `MAXDB` SQL mode enabled. When the server runs with this mode enabled, `TIMESTAMP` is identical with `DATETIME`. That is, if this mode is enabled at the time that a table is created, `TIMESTAMP` columns are created as `DATETIME` columns. As a result, such columns use `DATETIME` display format, have the same range of values, and there is no automatic initialization or updating to the current date and time.

To enable `MAXDB` mode, set the server SQL mode to `MAXDB` at startup using the `--sql-mode=MAXDB` server option or by setting the global `sql_mode` variable at runtime:

```
mysql> SET GLOBAL sql_mode=MAXDB;
```

A client can cause the server to run in `MAXDB` mode for its own connection as follows:

```
mysql> SET SESSION sql_mode=MAXDB;
```

10.3.2. The `TIME` Type

MySQL retrieves and displays `TIME` values in `'HH:MM:SS'` format (or `'HHH:MM:SS'` format for large hours values). `TIME` values may range from `'-838:59:59'` to `'838:59:59'`. The hours part may be so large because the `TIME` type can be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

You can specify `TIME` values in a variety of formats:

- As a string in `'D HH:MM:SS.fraction'` format. You can also use one of the following “relaxed” syntaxes: `'HH:MM:SS.fraction'`, `'HH:MM:SS'`, `'HH:MM'`, `'D HH:MM:SS'`, `'D HH:MM'`, `'D HH'`, or `'SS'`. Here `D` represents days and can have a value from 0 to 34. Note that MySQL does not store the fraction part.
- As a string with no delimiters in `'HHMMSS'` format, provided that it makes sense as a time. For example, `'101112'` is understood as `'10:11:12'`, but `'109712'` is illegal (it has a nonsensical minute part) and becomes `'00:00:00'`.
- As a number in `HHMMSS` format, provided that it makes sense as a time. For example, `101112` is understood as `'10:11:12'`. The following alternative formats are also understood: `SS`, `MMSS`, `HHMMSS`, `HHMMSS.fraction`. Note that MySQL does not store the fraction part.
- As the result of a function that returns a value that is acceptable in a `TIME` context, such as `CURRENT_TIME`.

A trailing `.uuuuuu` microseconds part of `TIME` values is allowed under the same conditions as for other temporal values, as described in [Section 10.3.1, “The DATETIME, DATE, and TIMESTAMP Types”](#). This includes the property that any microseconds part is discarded from values stored into `TIME` columns.

For `TIME` values specified as strings that include a time part delimiter, it is not necessary to specify two digits for hours, minutes, or seconds values that are less than 10. `'8:3:2'` is the same as `'08:03:02'`.

Be careful about assigning abbreviated values to a `TIME` column. Without colons, MySQL interprets values using the assumption that the two rightmost digits represent seconds. (MySQL interprets `TIME` values as elapsed time rather than as time of day.) For example, you might think of `'1112'` and `1112` as meaning `'11:12:00'` (12 minutes after 11 o'clock), but MySQL interprets them as `'00:11:12'` (11 minutes, 12 seconds). Similarly, `'12'` and `12` are interpreted as `'00:00:12'`. `TIME` values with colons, by con-

trast, are always treated as time of the day. That is, `'11:12'` mean `'11:12:00'`, not `'00:11:12'`.

By default, values that lie outside the `TIME` range but are otherwise legal are clipped to the closest endpoint of the range. For example, `'-850:00:00'` and `'850:00:00'` are converted to `'-838:59:59'` and `'838:59:59'`. Illegal `TIME` values are converted to `'00:00:00'`. Note that because `'00:00:00'` is itself a legal `TIME` value, there is no way to tell, from a value of `'00:00:00'` stored in a table, whether the original value was specified as `'00:00:00'` or whether it was illegal.

For more restrictive treatment of invalid `TIME` values, enable strict SQL mode to cause errors to occur. See [Section 5.2.6, “SQL Modes”](#).

`SERIAL` `DEFAULT` `VALUE` in the definition of an integer column is an alias for `NOT NULL AUTO_INCREMENT UNIQUE`.

10.3.3. The `YEAR` Type

The `YEAR` type is a one-byte type used for representing years.

MySQL retrieves and displays `YEAR` values in `YYYY` format. The range is 1901 to 2155.

You can specify `YEAR` values in a variety of formats:

- As a four-digit string in the range `'1901'` to `'2155'`.
- As a four-digit number in the range 1901 to 2155.
- As a two-digit string in the range `'00'` to `'99'`. Values in the ranges `'00'` to `'69'` and `'70'` to `'99'` are converted to `YEAR` values in the ranges 2000 to 2069 and 1970 to 1999.
- As a two-digit number in the range 1 to 99. Values in the ranges 1 to 69 and 70 to 99 are converted to `YEAR` values in the ranges 2001 to 2069 and 1970 to 1999. Note that the range for two-digit numbers is slightly different from the range for two-digit strings, because you cannot specify zero directly as a number and have it be interpreted as 2000. You must specify it as a string `'0'` or `'00'` or it is interpreted as 0000.
- As the result of a function that returns a value that is acceptable in a `YEAR` context, such as `NOW()`.

Illegal `YEAR` values are converted to 0000.

10.3.4. Year 2000 Issues and Date Types

MySQL Server itself has no problems with Year 2000 (Y2K) compliance:

- MySQL Server uses Unix time functions that handle dates into the year 2038 for `TIMESTAMP` values. For `DATE` and `DATETIME` values, dates through the year 9999 are accepted.
- All MySQL date functions are implemented in one source file, `sql/time.cc`, and are coded very carefully to be year 2000-safe.
- In MySQL, the `YEAR` data type can store the years 0 and 1901 to 2155 in one byte and display them using two or four digits. All two-digit years are considered to be in the range 1970 to 2069, which means that if you store 01 in a `YEAR` column, MySQL Server treats it as 2001.

Although MySQL Server itself is Y2K-safe, you may run into problems if you use it with applications that are not Y2K-safe. For example, many old applications store or manipulate years using two-digit values (which are ambiguous) rather than four-digit values. This problem may be compounded by applications that use values such as 00 or 99 as “missing” value indicators. Unfortunately, these problems may be difficult to fix because different applications may be written by different programmers, each of whom may use a different set of conventions and date-handling functions.

Thus, even though MySQL Server has no Y2K problems, *it is the application's responsibility to provide unambiguous input*. Any value containing a two-digit year is ambiguous, because the century is unknown. Such values must be interpreted into four-digit form because MySQL stores years internally using four digits.

For `DATETIME`, `DATE`, `TIMESTAMP`, and `YEAR` types, MySQL interprets dates with ambiguous year values using the following rules:

- Year values in the range **00-69** are converted to **2000-2069**.
- Year values in the range **70-99** are converted to **1970-1999**.

Remember that these rules are only heuristics that provide reasonable guesses as to what your data values mean. If the rules used by MySQL do not produce the correct values, you should provide unambiguous input containing four-digit year values.

ORDER BY properly sorts **YEAR** values that have two-digit years.

Some functions like **MIN()** and **MAX()** convert a **YEAR** to a number. This means that a value with a two-digit year does not work properly with these functions. The fix in this case is to convert the **TIMESTAMP** or **YEAR** to four-digit year format.

10.4. String Types

The string types are **CHAR**, **VARCHAR**, **BINARY**, **VARBINARY**, **BLOB**, **TEXT**, **ENUM**, and **SET**. This section describes how these types work and how to use them in your queries. For string type storage requirements, see [Section 10.5, “Data Type Storage Requirements”](#).

10.4.1. The **CHAR** and **VARCHAR** Types

The **CHAR** and **VARCHAR** types are similar, but differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained. No lettercase conversion takes place during storage or retrieval.

The **CHAR** and **VARCHAR** types are declared with a length that indicates the maximum number of characters you want to store. For example, **CHAR(30)** can hold up to 30 characters.

The length of a **CHAR** column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When **CHAR** values are stored, they are right-padded with spaces to the specified length. When **CHAR** values are retrieved, trailing spaces are removed.

Values in **VARCHAR** columns are variable-length strings. The length can be specified as a value from 0 to 65,535. (The maximum effective length of a **VARCHAR** is determined by the maximum row size and the character set used. The maximum column length is subject to a row size of 65,532 bytes.)

In contrast to **CHAR**, **VARCHAR** values are stored using only as many characters as are needed, plus one byte to record the length (two bytes for columns that are declared with a length longer than 255).

If strict SQL mode is not enabled and you assign a value to a **CHAR** or **VARCHAR** column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of non-space characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See [Section 5.2.6, “SQL Modes”](#).

VARCHAR values are not padded when they are stored. Trailing spaces are retained when values are stored and retrieved, in conformance with standard SQL.

The following table illustrates the differences between **CHAR** and **VARCHAR** by showing the result of storing various string values into **CHAR(4)** and **VARCHAR(4)** columns:

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
' '	' '	4 bytes	' '	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Note that the values shown as stored in the last row of the table apply *only when not using strict mode*; if MySQL is running in strict mode, values that exceed the column length are *not stored*, and an error results.

If a given value is stored into the **CHAR(4)** and **VARCHAR(4)** columns, the values retrieved from the columns are not always the same because trailing spaces are removed from **CHAR** columns upon retrieval. The following example illustrates this difference:

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> INSERT INTO vc VALUES ('ab ', 'ab ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT CONCAT('(', v, ')'), CONCAT('(', c, ')') FROM vc;
+-----+-----+
| CONCAT('(', v, ')') | CONCAT('(', c, ')') |
+-----+-----+
| (ab )              | (ab)                 |
+-----+-----+
1 row in set (0.06 sec)
```

Values in **CHAR** and **VARCHAR** columns are sorted and compared according to the character set collation assigned to the column.

Note that all MySQL collations are of type **PADSPACE**. This means that all **CHAR** and **VARCHAR** values in MySQL are compared without regard to any trailing spaces. For example:

```
mysql> CREATE TABLE names (myname CHAR(10), yourname VARCHAR(10));
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO names VALUES ('Monty ', 'Monty ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT myname = 'Monty ', yourname = 'Monty ' FROM names;
+-----+-----+
| myname = 'Monty ' | yourname = 'Monty ' |
+-----+-----+
| 1                  | 1                    |
+-----+-----+
1 row in set (0.00 sec)
```

Note that this is true for all MySQL versions, and that it is not affected by the server SQL mode.

For those cases where trailing pad characters are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad characters will result in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error.

10.4.2. The **BINARY** and **VARBINARY** Types

The **BINARY** and **VARBINARY** types are similar to **CHAR** and **VARCHAR**, except that they contain binary strings rather than non-binary strings. That is, they contain byte strings rather than character strings. This means that they have no character set, and sorting and comparison are based on the numeric values of the bytes in the values.

The allowable maximum length is the same for **BINARY** and **VARBINARY** as it is for **CHAR** and **VARCHAR**, except that the length for **BINARY** and **VARBINARY** is a length in bytes rather than in characters.

The **BINARY** and **VARBINARY** data types are distinct from the **CHAR BINARY** and **VARCHAR BINARY** data types. For the latter types, the **BINARY** attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary collation for the column character set to be used, and the column itself contains non-binary character strings rather than binary byte strings. For example, **CHAR(5) BINARY** is treated as **CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin**, assuming that the default character set is **latin1**. This differs from **BINARY(5)**, which stores 5-bytes binary strings that have no character set or collation.

If strict SQL mode is not enabled and you assign a value to a **BINARY** or **VARBINARY** column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For cases of truncation, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See [Section 5.2.6, “SQL Modes”](#).

When **BINARY** values are stored, they are right-padded with the pad value to the specified length. The pad value is **0x00** (the zero byte). Values are right-padded with **0x00** on insert, and no trailing bytes are removed on select. All bytes are significant in comparisons, including **ORDER BY** and **DISTINCT** operations. **0x00** bytes and spaces are different in comparisons, with **0x00** < space.

Example: For a **BINARY(3)** column, 'a ' becomes 'a \0' when inserted. 'a\0' becomes 'a\0\0' when inserted. Both inserted values remain unchanged when selected.

For **VARBINARY**, there is no padding on insert and no bytes are stripped on select. All bytes are significant in comparisons, including **ORDER BY** and **DISTINCT** operations. **0x00** bytes and spaces are different in comparisons, with **0x00** < space.

For those cases where trailing pad bytes are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad bytes will result in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a\0' causes a duplicate-key error.

You should consider the preceding padding and stripping characteristics carefully if you plan to use the **BINARY** data type for storing binary data and you require that the value retrieved be exactly the same as the value stored. The following example illustrates how **0x00**-padding of **BINARY** values affects column value comparisons:

```
mysql> CREATE TABLE t (c BINARY(3));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET c = 'a';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a\0\0' from t;
+-----+-----+-----+
| HEX(c) | c = 'a' | c = 'a\0\0' |
+-----+-----+-----+
| 610000 | 0       | 1           |
+-----+-----+-----+
1 row in set (0.09 sec)
```

If the value retrieved must be the same as the value specified for storage with no padding, it might be preferable to use **VARBINARY** or one of the **BLOB** data types instead.

10.4.3. The **BLOB** and **TEXT** Types

A **BLOB** is a binary large object that can hold a variable amount of data. The four **BLOB** types are **TINYBLOB**, **BLOB**, **MEDIUMBLOB**, and **LONGBLOB**. These differ only in the maximum length of the values they can hold. The four **TEXT** types are **TINYTEXT**, **TEXT**, **MEDIUMTEXT**, and **LONGTEXT**. These correspond to the four **BLOB** types and have the same maximum lengths and storage requirements. See [Section 10.5, “Data Type Storage Requirements”](#). No lettercase conversion for **TEXT** or **BLOB** columns takes place during storage or retrieval.

BLOB columns are treated as binary strings (byte strings). **TEXT** columns are treated as non-binary strings (character strings). **BLOB** columns have no character set, and sorting and comparison are based on the numeric values of the bytes in column values. **TEXT** columns have a character set, and values are sorted and compared based on the collation of the character set.

If strict SQL mode is not enabled and you assign a value to a **BLOB** or **TEXT** column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of non-space characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See [Section 5.2.6, “SQL Modes”](#).

If a **TEXT** column is indexed, index entry comparisons are space-padded at the end. This means that, if the index requires unique values, duplicate-key errors will occur for values that differ only in the number of trailing spaces. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error. This is not true for **BLOB** columns.

In most respects, you can regard a **BLOB** column as a **VARBINARY** column that can be as large as you like. Similarly, you can regard a **TEXT** column as a **VARCHAR** column. **BLOB** and **TEXT** differ from **VARBINARY** and **VARCHAR** in the following ways:

- For indexes on **BLOB** and **TEXT** columns, you must specify an index prefix length. For **CHAR** and **VARCHAR**, a prefix length is optional. See [Section 6.4.3, “Column Indexes”](#).
- BLOB** and **TEXT** columns cannot have **DEFAULT** values.

LONG and **LONG VARCHAR** map to the **MEDIUMTEXT** data type. This is a compatibility feature. If you use the **BINARY** attribute with a **TEXT** data type, the column is assigned the binary collation of the column character set.

MySQL Connector/ODBC defines **BLOB** values as **LONGVARBINARY** and **TEXT** values as **LONGVARCHAR**.

Because **BLOB** and **TEXT** values can be extremely long, you might encounter some constraints in using them:

- Only the first **max_sort_length** bytes of the column are used when sorting. The default value of **max_sort_length** is 1024. This value can be changed using the **--max_sort_length=N** option when starting the **mysqld** server. See [Section 5.2.3, “System Variables”](#).

You can make more bytes significant in sorting or grouping by increasing the value of **max_sort_length** at runtime. Any client can change the value of its session **max_sort_length** variable:

```
mysql> SET max_sort_length = 2000;
mysql> SELECT id, comment FROM t
-> ORDER BY comment;
```

Another way to use `GROUP BY` or `ORDER BY` on a `BLOB` or `TEXT` column containing long values when you want more than `max_sort_length` bytes to be significant is to convert the column value into a fixed-length object. The standard way to do this is with the `SUBSTRING()` function. For example, the following statement causes 2000 bytes of the `comment` column to be taken into account for sorting:

```
mysql> SELECT id, SUBSTRING(comment,1,2000) FROM t
-> ORDER BY SUBSTRING(comment,1,2000);
```

- The maximum size of a `BLOB` or `TEXT` object is determined by its type, but the largest value you actually can transmit between the client and server is determined by the amount of available memory and the size of the communications buffers. You can change the message buffer size by changing the value of the `max_allowed_packet` variable, but you must do so for both the server and your client program. For example, both `mysql` and `mysqldump` allow you to change the client-side `max_allowed_packet` value. See [Section 6.5.2, “Tuning Server Parameters”](#), [Section 7.8, “mysql — The MySQL Command-Line Tool”](#), and [Section 7.13, “mysqldump — A Database Backup Program”](#). You may also want to compare the packet sizes and the size of the data objects you are storing with the storage requirements, see [Section 10.5, “Data Type Storage Requirements”](#)

Each `BLOB` or `TEXT` value is represented internally by a separately allocated object. This is in contrast to all other data types, for which storage is allocated once per column when the table is opened.

In some cases, it may be desirable to store binary data such as media files in `BLOB` or `TEXT` columns. You may find MySQL's string handling functions useful for working with such data. See [Section 11.4, “String Functions”](#). For security and other reasons, it is usually preferable to do so using application code rather than allowing application users the `FILE` privilege. You can discuss specifics for various languages and platforms in the MySQL Forums (<http://forums.mysql.com/>).

10.4.4. The `ENUM` Type

An `ENUM` is a string object with a value chosen from a list of allowed values that are enumerated explicitly in the column specification at table creation time.

An enumeration value must be a quoted string literal; it may not be an expression, even one that evaluates to a string value. This means that you also may not employ a user variable as an enumeration value.

The value may also be the empty string (`' '`) or `NULL` under certain circumstances:

- If you insert an invalid value into an `ENUM` (that is, a string not present in the list of allowed values), the empty string is inserted instead as a special error value. This string can be distinguished from a “normal” empty string by the fact that this string has the numerical value 0. More about this later.

If strict SQL mode is enabled, attempts to insert invalid `ENUM` values result in an error.

- If an `ENUM` column is declared to allow `NULL`, the `NULL` value is a legal value for the column, and the default value is `NULL`. If an `ENUM` column is declared `NOT NULL`, its default value is the first element of the list of allowed values.

Each enumeration value has an index:

- Values from the list of allowable elements in the column specification are numbered beginning with 1.
- The index value of the empty string error value is 0. This means that you can use the following `SELECT` statement to find rows into which invalid `ENUM` values were assigned:

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- The index of the `NULL` value is `NULL`.
- The term “index” here refers only to position within the list of enumeration values. It has nothing to do with table indexes.

For example, a column specified as `ENUM('one', 'two', 'three')` can have any of the values shown here. The index of each value is also shown:

Value	Index
NULL	NULL
' '	0
'one'	1
'two'	2
'three'	3

An enumeration can have a maximum of 65,535 elements.

Trailing spaces are automatically deleted from **ENUM** member values in the table definition when a table is created.

When retrieved, values stored into an **ENUM** column are displayed using the lettercase that was used in the column definition. Note that **ENUM** columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

If you retrieve an **ENUM** value in a numeric context, the column value's index is returned. For example, you can retrieve numeric values from an **ENUM** column like this:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

If you store a number into an **ENUM** column, the number is treated as the index into the possible values, and the value stored is the enumeration member with that index. (However, this does *not* work with **LOAD DATA**, which treats all input as strings.) If the numeric value is quoted, it is still interpreted as an index if there is no matching string in the list of enumeration values. For these reasons, it is not advisable to define an **ENUM** column with enumeration values that look like numbers, because this can easily become confusing. For example, the following column has enumeration members with string values of '0', '1', and '2', but numeric index values of 1, 2, and 3:

```
numbers ENUM('0', '1', '2')
```

If you store 2, it is interpreted as an index value, and becomes '1' (the value with index 2). If you store '2', it matches an enumeration value, so it is stored as '2'. If you store '3', it does not match any enumeration value, so it is treated as an index and becomes '2' (the value with index 3).

```
mysql> INSERT INTO t (numbers) VALUES(2),('2'),('3');
mysql> SELECT * FROM t;
+-----+
| numbers |
+-----+
| 1       |
| 2       |
| 2       |
+-----+
```

ENUM values are sorted according to the order in which the enumeration members were listed in the column specification. (In other words, **ENUM** values are sorted according to their index numbers.) For example, 'a' sorts before 'b' for **ENUM('a', 'b')**, but 'b' sorts before 'a' for **ENUM('b', 'a')**. The empty string sorts before non-empty strings, and **NULL** values sort before all other enumeration values. To prevent unexpected results, specify the **ENUM** list in alphabetical order. You can also use **GROUP BY CAST(col AS CHAR)** or **GROUP BY CONCAT(col)** to make sure that the column is sorted lexically rather than by index number.

Functions such as **SUM()** or **AVG()** that expect a numeric argument cast the argument to a number if necessary. For **ENUM** values, the cast operation causes the index number to be used.

If you want to determine all possible values for an **ENUM** column, use **SHOW COLUMNS FROM tbl_name LIKE enum_col** and parse the **ENUM** definition in the **Type** column of the output.

10.4.5. The SET Type

A **SET** is a string object that can have zero or more values, each of which must be chosen from a list of allowed values specified when the table is created. **SET** column values that consist of multiple set members are specified with members separated by commas (','). A consequence of this is that **SET** member values should not themselves contain commas.

For example, a column specified as **SET('one', 'two') NOT NULL** can have any of these values:

```
'one'
'two'
'one, two'
```

A **SET** can have a maximum of 64 different members.

Trailing spaces are automatically deleted from **SET** member values in the table definition when a table is created.

When retrieved, values stored in a **SET** column are displayed using the lettercase that was used in the column definition. Note that **SET** columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

MySQL stores **SET** values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a **SET** value in a numeric context, the value retrieved has bits set corresponding to the set members that make up the column value. For example, you can retrieve numeric values from a **SET** column like this:

```
mysql> SELECT set_col+0 FROM tbl_name;
```

If a number is stored into a **SET** column, the bits that are set in the binary representation of the number determine the set members in the column value. For a column specified as **SET('a', 'b', 'c', 'd')**, the members have the following decimal and binary values:

SET Member	Decimal Value	Binary Value
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

If you assign a value of 9 to this column, that is 1001 in binary, so the first and fourth **SET** value members 'a' and 'd' are selected and the resulting value is 'a,d'.

For a value containing more than one **SET** element, it does not matter what order the elements are listed in when you insert the value. It also does not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value appears once, with elements listed according to the order in which they were specified at table creation time. For example, suppose that a column is specified as **SET('a', 'b', 'c', 'd')**:

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

If you insert the values 'a,d', 'd,a', 'a,d,d', 'a,d,a', and 'd,a,d':

```
mysql> INSERT INTO myset (col) VALUES
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
Query OK, 5 rows affected (0.01 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

Then all of these values appear as 'a,d' when retrieved:

```
mysql> SELECT col FROM myset;
+-----+
| col   |
+-----+
| a,d   |
| a,d   |
| a,d   |
| a,d   |
| a,d   |
+-----+
5 rows in set (0.04 sec)
```

If you set a **SET** column to an unsupported value, the value is ignored and a warning is issued:

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
|
```



```

| Level | Code | Message |
+-----+-----+-----+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+-----+-----+-----+
1 row in set (0.04 sec)

mysql> SELECT col FROM myset;
+-----+
| col |
+-----+
| a,d |
| a,d |
| a,d |
| a,d |
| a,d |
| a,d |
+-----+
6 rows in set (0.01 sec)

```

If strict SQL mode is enabled, attempts to insert invalid [SET](#) values result in an error.

[SET](#) values are sorted numerically. [NULL](#) values sort before non-[NULL SET](#) values.

Functions such as [SUM\(\)](#) or [AVG\(\)](#) that expect a numeric argument cast the argument to a number if necessary. For [SET](#) values, the cast operation causes the numeric value to be used.

Normally, you search for [SET](#) values using the [FIND_IN_SET\(\)](#) function or the [LIKE](#) operator:

```

mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';

```

The first statement finds rows where [set_col](#) contains the [value](#) set member. The second is similar, but not the same: It finds rows where [set_col](#) contains [value](#) anywhere, even as a substring of another set member.

The following statements also are legal:

```

mysql> SELECT * FROM tbl_name WHERE set_col & 1;
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';

```

The first of these statements looks for values containing the first set member. The second looks for an exact match. Be careful with comparisons of the second type. Comparing set values to ['val1,val2'](#) returns different results than comparing values to ['val2,val1'](#). You should specify the values in the same order they are listed in the column definition.

If you want to determine all possible values for a [SET](#) column, use [SHOW COLUMNS FROM tbl_name LIKE set_col](#) and parse the [SET](#) definition in the [Type](#) column of the output.

10.5. Data Type Storage Requirements

The storage requirements for each of the data types supported by MySQL are listed here by category.

The maximum size of a row in a [MyISAM](#) table is 65,534 bytes. Each [BLOB](#) and [TEXT](#) column accounts for only five to nine bytes toward this size. This limitation may be shared by other storage engines as well.

Important

For tables using the [NDBCluster](#) storage engine, there is the factor of *4-byte alignment* to be taken into account when calculating storage requirements. This means that all [NDB](#) data storage is done in multiples of 4 bytes. Thus, a column value that would take 15 bytes in a table using a storage engine other than [NDB](#) requires 16 bytes in an [NDB](#) table. This requirement applies in addition to any other considerations that are discussed in this section. For example, in [NDBCluster](#) tables, the [TINYINT](#), [SMALLINT](#), [MEDIUMINT](#), and [INTEGER \(INT\)](#) column types each require 4 bytes storage per record due to the alignment factor.

In addition, when calculating storage requirements for Cluster tables, you must remember that every table using the [NDBCluster](#) storage engine requires a primary key; if no primary key is defined by the user, then a “hidden” primary key will be created by [NDB](#). This hidden primary key consumes 31-35 bytes per table record.

You may find the [ndb_size.pl](#) utility to be useful for estimating [NDB](#) storage requirements. This Perl script connects to a current MySQL (non-Cluster) database and creates a report on how much space that database would require if it used the [NDBCluster](#) storage engine. See [Section 16.9.15, “ndb_size.pl — NDBCluster Size Requirement Estimator”](#), for more information.

Storage Requirements for Numeric Types

Data Type	Storage Required
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT , INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(<i>p</i>)	4 bytes if $0 \leq p \leq 24$, 8 bytes if $25 \leq p \leq 53$
FLOAT	4 bytes
DOUBLE [PRECISION] , REAL	8 bytes
DECIMAL(<i>M</i>, <i>D</i>) , NUMERIC(<i>M</i>, <i>D</i>)	Varies; see following discussion
BIT(<i>M</i>)	approximately $(M+7)/8$ bytes

Values for [DECIMAL](#) (and [NUMERIC](#)) columns are represented using a binary format that packs nine decimal (base 10) digits into four bytes. Storage for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires four bytes, and the “leftover” digits require some fraction of four bytes. The storage required for excess digits is given by the following table:

Leftover Digits	Number of Bytes
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4

Storage Requirements for Date and Time Types

Data Type	Storage Required
DATE	3 bytes
TIME	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
YEAR	1 byte

The storage requirements shown in the table arise from the way that MySQL represents temporal values:

- [DATE](#): A three-byte integer packed as $DD + MM \times 32 + YYYY \times 16 \times 32$
- [TIME](#): A three-byte integer packed as $DD \times 24 \times 3600 + HH \times 3600 + MM \times 60 + SS$
- [DATETIME](#): Eight bytes:
 - A four-byte integer packed as $YYYY \times 10000 + MM \times 100 + DD$
 - A four-byte integer packed as $HH \times 10000 + MM \times 100 + SS$

- **TIMESTAMP**: A four-byte integer representing seconds UTC since the epoch ('1970-01-01 00:00:00' UTC)
- **YEAR**: A one-byte integer

Storage Requirements for String Types

Data Type	Storage Required
CHAR (<i>M</i>)	<i>M</i> characters, $0 \leq M \leq 255$
VARCHAR (<i>M</i>)	<i>L</i> characters + 1 byte, where $L \leq M$ and $0 \leq M \leq 255$ or <i>L</i> characters + 2 bytes, where $L \leq M$ and $256 \leq M \leq 65535$
BINARY (<i>M</i>)	<i>M</i> bytes, $0 \leq M \leq 255$
VARBINARY (<i>M</i>)	<i>L</i> + 1 bytes, where $L \leq M$ and $0 \leq M \leq 255$ or <i>L</i> + 2 bytes, where $L \leq M$ and $256 \leq M \leq 65535$
TINYBLOB	<i>L</i> + 1 bytes, where $L < 2^8$
TINYTEXT	<i>L</i> characters + 1 byte, where $L < 2^8$
BLOB	<i>L</i> + 2 bytes, where $L < 2^{16}$
TEXT	<i>L</i> characters + 2 bytes, where $L < 2^{16}$
MEDIUMBLOB	<i>L</i> + 3 bytes, where $L < 2^{24}$
MEDIUMTEXT	<i>L</i> characters + 3 bytes, where $L < 2^{24}$
LOBLOB	<i>L</i> + 4 bytes, where $L < 2^{32}$
LONGTEXT	<i>L</i> characters + 4 bytes, where $L < 2^{32}$
ENUM ('value1', 'value2', ...)	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
SET ('value1', 'value2', ...)	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

For the **CHAR**, **VARCHAR**, and **TEXT** types, the values *L* and *M* in the preceding table should be interpreted as number of characters, and lengths for these types in column specifications indicate the number of characters. For example, to store a **TINYTEXT** value requires *L* characters to store the value plus one byte to store the length of the value.

To calculate the number of *bytes* used to store a particular **CHAR**, **VARCHAR**, or **TEXT** column value, you must take into account the character set used for that column and whether the value contains multi-byte characters. In particular, when using the **utf8** Unicode character set, you must keep in mind that not all **utf8** characters use the same number of bytes and can require up to three bytes per character. For a breakdown of the storage used for different categories of **utf8** characters, see [Section 9.8, “Unicode Support”](#).

VARCHAR, **VARBINARY**, and the **BLOB** and **TEXT** types are variable-length types. For each, the storage requirements depend on these factors:

- The actual length of the column value
- The column's maximum possible length
- The character set used for the column, because some character sets contain multi-byte characters

For example, a **VARCHAR**(10) column can hold a string with a maximum length of 10 characters. Assuming that the column uses the **latin1** character set (one byte per character), the actual storage required is the length of the string (*L*), plus one byte to record the length of the string. For the string 'abcd', *L* is 4 and the storage requirement is five bytes. If the same column was instead declared as **VARCHAR**(500), the string 'abcd' requires $4 + 2 = 6$ bytes. Two bytes rather than one are required to store the column length because the maximum length is greater than 255 characters.

Note

The effective maximum number of *bytes* that can be stored in a **VARCHAR** or **VARBINARY** column is 65,532. For a **VARCHAR** column that stores multi-byte characters, the maximum effective number of *characters* is less. For example, **utf8** characters can require up to three bytes per character, so a **VARCHAR** column that uses the **utf8** character set can

■ be declared to be a maximum of 21,844 characters.

The **NDBCLUSTER** storage engine in MySQL 5.1 supports variable-width columns. This means that a **VARCHAR** column in a MySQL Cluster table requires the same amount of storage as it would using any other storage engine, with the exception that such values are 4-byte aligned. Thus, the string 'abcd' stored in a **VARCHAR(50)** column using the **latin1** character set requires 8 bytes (rather than 6 bytes for the same column value in a **MyISAM** table). This represents a change in behavior from earlier versions of **NDB-CLUSTER**, where a **VARCHAR(50)** column would require 52 bytes storage per record regardless of the length of the string being stored.

The **BLOB** and **TEXT** types require 1, 2, 3, or 4 bytes to record the length of the column value, depending on the maximum possible length of the type. See [Section 10.4.3, “The BLOB and TEXT Types”](#).

TEXT and **BLOB** columns are implemented differently in the NDB Cluster storage engine, wherein each row in a **TEXT** column is made up of two separate parts. One of these is of fixed size (256 bytes), and is actually stored in the original table. The other consists of any data in excess of 256 bytes, which is stored in a hidden table. The rows in this second table are always 2,000 bytes long. This means that the size of a **TEXT** column is 256 if *size* ≤ 256 (where *size* represents the size of the row); otherwise, the size is 256 + *size* + (2000 – (*size* – 256) % 2000).

The size of an **ENUM** object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations having between 256 and 65,535 possible values. See [Section 10.4.4, “The ENUM Type”](#).

The size of a **SET** object is determined by the number of different set members. If the set size is *N*, the object occupies (*N*+7)/8 bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A **SET** can have a maximum of 64 members. See [Section 10.4.5, “The SET Type”](#).

10.6. Choosing the Right Type for a Column

For optimum storage, you should try to use the most precise type in all cases. For example, if an integer column is used for values in the range from 1 to 99999, **MEDIUMINT UNSIGNED** is the best type. Of the types that represent all the required values, this type uses the least amount of storage.

All basic calculations (+, -, *, /) with **DECIMAL** columns are done with precision of 65 decimal (base 10) digits. See [Section 10.1.1, “Overview of Numeric Types”](#).

If accuracy is not too important or if speed is the highest priority, the **DOUBLE** type may be good enough. For high precision, you can always convert to a fixed-point type stored in a **BIGINT**. This allows you to do all calculations with 64-bit integers and then convert results back to floating-point values as necessary.

10.7. Using Data Types from Other Database Engines

To facilitate the use of code written for SQL implementations from other vendors, MySQL maps data types as shown in the following table. These mappings make it easier to import table definitions from other database systems into MySQL:

Other Vendor Type	MySQL Type
BOOL	TINYINT
BOOLEAN	TINYINT
CHARACTER VARYING(M)	VARCHAR(M)
FIXED	DECIMAL
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT

MIDDLEINT	MEDIUMINT
NUMERIC	DECIMAL

Data type mapping occurs at table creation time, after which the original type specifications are discarded. If you create a table with types used by other vendors and then issue a `DESCRIBE tbl_name` statement, MySQL reports the table structure using the equivalent MySQL types. For example:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DESCRIBE t;
```

Field	Type	Null	Key	Default	Extra
a	tinyint(1)	YES		NULL	
b	double	YES		NULL	
c	mediumtext	YES		NULL	
d	decimal(10,0)	YES		NULL	

```
4 rows in set (0.01 sec)
```

Chapter 11. Functions and Operators

Expressions can be used at several points in SQL statements, such as in the **ORDER BY** or **HAVING** clauses of **SELECT** statements, in the **WHERE** clause of a **SELECT**, **DELETE**, or **UPDATE** statement, or in **SET** statements. Expressions can be written using literal values, column values, **NULL**, built-in functions, stored functions, user-defined functions, and operators. This chapter describes the functions and operators that are allowed for writing expressions in MySQL. Instructions for writing stored functions and user-defined functions are given in [Chapter 19, *Stored Procedures and Functions*](#), and [Section 28.3, “Adding New Functions to MySQL”](#). See [Section 8.2.4, “Function Name Parsing and Resolution”](#), for the rules describing how the server interprets references to different kinds of functions.

An expression that contains **NULL** always produces a **NULL** value unless otherwise indicated in the documentation for a particular function or operator.

Note

By default, there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function calls and references to tables or columns that happen to have the same name as a function. However, spaces around function arguments are permitted.

You can tell the MySQL server to accept spaces after function names by starting it with the `--sql-mode=IGNORE_SPACE` option. (See [Section 5.2.6, “SQL Modes”](#).) Individual client programs can request this behavior by using the `CLIENT_IGNORE_SPACE` option for `mysql_real_connect()`. In either case, all function names become reserved words.

For the sake of brevity, most examples in this chapter display the output from the `mysql` program in abbreviated form. Rather than showing examples in this format:

```
mysql> SELECT MOD(29,9);
+-----+
| mod(29,9) |
+-----+
|          2 |
+-----+
1 rows in set (0.00 sec)
```

This format is used instead:

```
mysql> SELECT MOD(29,9);
-> 2
```

11.1. Operator and Function Reference

Note

This table is part of an ongoing process to expand and simplify the information provided on these elements. Further improvements to the table, and corresponding descriptions will be applied over the coming months.

Name	Description
ABS()	Return the absolute value
ACOS()	Return the arc cosine
ADDDATE() (v4.1.1)	Add dates
ADDTIME() (v4.1.1)	Add time
AES_DECRYPT()	Decrypt using AES
AES_ENCRYPT()	Encrypt using AES
AND, &&	Logical AND
ASCII()	Return numeric value of left-most character
ASIN()	Return the arc sine
ATAN2() , ATAN()	Return the arc tangent of the two arguments
ATAN()	Return the arc tangent
AVG()	Return the average value of the argument

Name	Description
BENCHMARK()	Repeatedly execute an expression
BETWEEN ... AND ...	Check whether a value is within a range of values
BIN()	Return a string representation of the argument
BINARY	Cast a string to a binary string
BIT_AND()	Return bitwise and
BIT_COUNT()	Return the number of bits that are set
BIT_LENGTH()	Return length of argument in bits
BIT_OR()	Return bitwise or
BIT_XOR() (v4.1.1)	Return bitwise xor
&	Bitwise AND
 	Bitwise OR
^	Bitwise XOR
/	Division operator
CASE	Case statement
CAST()	Cast a value as a certain type
CEILING() , CEIL()	Return the smallest integer value not less than the argument
CHAR_LENGTH()	Return number of characters in argument
CHAR()	Return the character for each integer passed
CHARACTER_LENGTH()	A synonym for CHAR_LENGTH()
CHARSET() (v4.1.0)	Return the character set of the argument
COALESCE()	Return the first non-NULL argument
COERCIBILITY() (v4.1.1)	Return the collation coercibility value of the string argument
COLLATION() (v4.1.0)	Return the collation of the string argument
COMPRESS() (v4.1.1)	Return result as a binary string
CONCAT_WS()	Return concatenate with separator
CONCAT()	Return concatenated string
CONNECTION_ID()	Return the connection ID (thread ID) for the connection
CONV()	Convert numbers between different number bases
CONVERT_TZ() (v4.1.3)	Convert from one timezone to another
COS()	Return the cosine
COT()	Return the cotangent
COUNT(DISTINCT)	Return the count of a number of different values
COUNT()	Return a count of the number of rows returned
CRC32() (v4.1.0)	Compute a cyclic redundancy check value
CURDATE()	Return the current date
CURRENT_DATE() , CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME() , CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP() , CURRENT_TIMESTAMP	Synonyms for NOW()
CURRENT_USER() , CURRENT_USER	Return the username and hostname combination
CURTIME()	Return the current time
DATABASE()	Return the default (current) database name
DATE_ADD()	Add two dates
DATE_FORMAT()	Format date as specified

Name	Description
DATE_SUB()	Subtract two dates
DATE()(v4.1.1)	Extract the date part of a date or datetime expression
DATEDIFF()(v4.1.1)	Subtract two dates
DAY()(v4.1.1)	Synonym for DAYOFMONTH()
DAYNAME()(v4.1.21)	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (1-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
DECODE()	Decodes a string encrypted using ENCODE()
DEFAULT()	Return the default value for a table column
DEGREES()	Convert radians to degrees
DES_DECRYPT()	Decrypt a string
DES_ENCRYPT()	Decrypt a string
DIV(v4.1.0)	Integer division
ELT()	Return string at index number
ENCODE()	Encode a string
ENCRYPT()	Encrypt a string
<=>	NULL-safe equal to operator
=	Equal operator
EXP()	Raise to the power of
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
EXTRACT	Extract part of a date
ExtractValue()(v5.1.5)	Extracts a value from an XML string using XPath notation
FIELD()	Return the index (position) of the first argument in the subsequent arguments
FIND_IN_SET()	Return the index position of the first argument within the second argument
FLOOR()	Return the largest integer value not greater than the argument
FORMAT()	Return a number formatted to specified number of decimal places
FOUND_ROWS()	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format date as a UNIX timestamp
MOD()	Return the remainder
GET_FORMAT()(v4.1.1)	Return a date format string
GET_LOCK()	Get a named lock
>=	Greater than or equal operator
>	Greater than operator
GREATEST()	Return the largest argument
GROUP_CONCAT()(v4.1)	Return a concatenated string
HEX()	Return a string representation of a hex value
HOUR()	Extract the hour
IF()	If/else construct
IFNULL()	Null if/else construct
IN	Check whether a value is within a set of values
INET_ATON()	Return the numeric value of an IP address

Name	Description
INET_NTOA()	Return the IP address from a numeric value
INSERT()	Insert a substring at the specified position up to the specified number of characters
INSTR()	Return the index of the first occurrence of substring
INTERVAL()	Return the index of the argument that is less than the first argument
IS_FREE_LOCK()	Checks whether the named lock is free
IS NULL	NULL value test
IS_USED_LOCK() (v4.1.0)	Checks whether the named lock is in use. Return connection identifier if true.
IS	Test a value against a boolean
ISNULL()	Test whether the argument is NULL
LAST_DAY (v4.1.1)	Return the last day of the month for the argument
LAST_INSERT_ID()	Value of the AUTOINCREMENT column for the last INSERT
LCASE()	Synonym for LOWER()
LEAST()	Return the smallest argument
<<	Left shift
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
<=	Less than or equal operator
<	Less than operator
LIKE	Simple pattern matching
LN()	Return the natural logarithm of the argument
LOAD_FILE()	Load the named file
LOCALTIME() , LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP , LOCALTIMESTAMP() (v4.0.6)	Synonym for NOW()
LOCATE()	Return the position of the first occurrence of substring
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MAKEDATE() (v4.1.1)	Create a date from the year and day of year
MAKETIME (v4.1.1)	MAKETIME()
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
MAX()	Return the maximum value
MD5()	Calculate MD5 checksum
MICROSECOND() (v4.1.1)	Return the microseconds from argument
MID()	Return a substring starting from the specified position
MIN()	Return the minimum value
-	Minus operator
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed

Name	Description
MONTHNAME() (v4.1.21)	Return the name of the month
NAME_CONST() (v5.0.12)	Causes the column to have the given name
NOT BETWEEN ... AND ...	Check whether a value is not within a range of values
!=, <>	Not equal operator
NOT IN	Check whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
NOT, !	Negates value
NOW()	Return the current date and time
NULLIF()	Return NULL if expr1 = expr2
OCT()	Return a string representation of the octal argument
OCTET_LENGTH()	A synonym for LENGTH()
OLD_PASSWORD() (v4.1)	Return the value of the old (pre-4.1) implementation of PASSWORD
%	Modulo operator
, OR	Logical OR
ORD()	If the leftmost character of the argument is a multi-byte character, returns the code for that character
PASSWORD()	Calculate and return a password string
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
PI()	Return the value of pi
+	Addition operator
POSITION()	A synonym for LOCATE()
POW(), POWER()	Return the argument raised to the specified power
PROCEDURE ANALYSE()	Analyze the results of a query
QUARTER()	Return the quarter from a date argument
QUOTE()	Escape the argument for use in an SQL statement
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
REGEXP	Pattern matching using regular expressions
RELEASE_LOCK()	Releases the named lock
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
>>	Right shift
RIGHT()	Return the specified rightmost number of characters
RLIKE	Synonym for REGEXP
ROUND()	Round the argument
ROW_COUNT() (v5.0.1)	The number of rows updated
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SCHEMA() (v5.0.2)	A synonym for DATABASE()
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)

Name	Description
SESSION_USER()	Synonym for USER()
SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SLEEP()(v5.0.12)	Sleep for a number of seconds
SOUNDEX()	Return a soundex string
SOUNDS LIKE(v4.1.0)	Compare sounds
SPACE()	Return a string of the specified number of spaces
SQRT()	Return the square root of the argument
STD(), STDDEV()	Return the population standard deviation
STDDEV_POP()(v5.0.3)	Return the population standard deviation
STDDEV_SAMP()(v5.0.3)	Return the sample standard deviation
STR_TO_DATE()(v4.1.1)	Convert a string to a date
STRCMP()	Compare two strings
SUBDATE()	When invoked with three arguments a synonym for DATE_SUB()
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING(), SUBSTR()	Return the substring as specified
SUBTIME()(v4.1.1)	Subtract times
SUM()	Return the sum
SYSDATE()	Return the time at which the function executes
SYSTEM_USER()	Synonym for USER()
TAN()	Return the tangent of the argument
~	Invert bits
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIME()(v4.1.1)	Extract the time portion of the expression passed
TIMEDIFF()(v4.1.1)	Subtract time
*	Times operator
TIMESTAMP()(v4.1.1)	With a single argument, this function returns the date or datetime expression. With two arguments, the sum of the arguments
TIMESTAMPADD()(v5.0.0)	Add an interval to a datetime expression
TIMESTAMPDIFF()(v5.0.0)	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
TRIM()	Remove leading and trailing spaces
TRUNCATE()	Truncate to specified number of decimal places
UCASE()	Synonym for UPPER()
-	Change the sign of the argument
UNCOMPRESS()(v4.1.1)	Uncompress a string compressed
UNCOMPRESSED_LENGTH()(v4.1.1)	Return the length of a string before compression
UNHEX()(v4.1.2)	Convert each pair of hexadecimal digits to a character
UNIX_TIMESTAMP()	Return a UNIX timestamp
UpdateXML()(v5.1.5)	Return replaced XML fragment
UPPER()	Convert to uppercase
USER()	Return the current username and hostname

Name	Description
UTC_DATE() (v4.1.1)	Return the current UTC date
UTC_TIME() (v4.1.1)	Return the current UTC time
UTC_TIMESTAMP() (v4.1.1)	Return the current UTC date and time
UUID() (v4.1.2)	Return a Universal Unique Identifier (UUID)
VALUES() (v4.1.1)	Defines the values to be used during an INSERT
VAR_POP() (v5.0.3)	Return the population standard variance
VAR_SAMP() (v5.0.3)	Return the sample variance
VARIANCE() (v4.1)	Return the population standard variance
VERSION()	Returns a string that indicates the MySQL server version
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR() (v4.1.1)	Return the calendar week of the date (1-53)
XOR	Logical XOR
YEAR()	Return the year
YEARWEEK()	Return the year and week

11.2. Operators

Name	Description
AND, &&	Logical AND
BINARY	Cast a string to a binary string
&	Bitwise AND
 	Bitwise OR
^	Bitwise XOR
/	Division operator
DIV (v4.1.0)	Integer division
<=>	NULL-safe equal to operator
=	Equal operator
>=	Greater than or equal operator
>	Greater than operator
IS NULL	NULL value test
IS	Test a value against a boolean
<<	Left shift
<=	Less than or equal operator
<	Less than operator
LIKE	Simple pattern matching
-	Minus operator
!=, <>	Not equal operator
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
NOT, !	Negates value
%	Modulo operator
 , OR	Logical OR
+	Addition operator

Name	Description
REGEXP	Pattern matching using regular expressions
>>	Right shift
RLIKE	Synonym for REGEXP
SOUNDS LIKE (v4.1.0)	Compare sounds
~	Invert bits
*	Times operator
-	Change the sign of the argument
XOR	Logical XOR

11.2.1. Operator Precedence

Operator precedences are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

```
BINARY, COLLATE
!
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
&&, AND
XOR
||, OR
:=
```

The `||` operator has a precedence between `^` and the unary operators if the `PIPES_AS_CONCAT` SQL mode is enabled.

Note

If the `HIGH_NOT_PRECEDENCE` SQL mode is enabled, the precedence of `NOT` is the same as that of the `!` operator. See [Section 5.2.6, “SQL Modes”](#).

The precedence of operators determines the order of evaluation of terms in an expression. To override this order and group terms explicitly, use parentheses. For example:

```
mysql> SELECT 1+2*3;
-> 7
mysql> SELECT (1+2)*3;
-> 9
```

11.2.2. Type Conversion in Expression Evaluation

When an operator is used with operands of different types, type conversion occurs to make the operands compatible. Some conversions occur implicitly. For example, MySQL automatically converts numbers to strings as necessary, and vice versa.

```
mysql> SELECT 1+'1';
-> 2
mysql> SELECT CONCAT(2, ' test');
-> '2 test'
```

It is also possible to perform explicit conversions. If you want to convert a number to a string explicitly, use the `CAST()` or `CONCAT()` function (`CAST()` is preferable):

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
-> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
-> 38.8, '38.8'
```

The following rules describe how conversion occurs for comparison operations:

- If one or both arguments are **NULL**, the result of the comparison is **NULL**, except for the **NULL**-safe **<=>** equality comparison operator. For **NULL <=> NULL**, the result is true.
- If both arguments in a comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- Hexadecimal values are treated as binary strings if not compared to a number.
- If one of the arguments is a **TIMESTAMP** or **DATETIME** column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more ODBC-friendly. Note that this is not done for the arguments to **IN()**! To be safe, always use complete datetime, date, or time strings when doing comparisons.
- In all other cases, the arguments are compared as floating-point (real) numbers.

The following examples illustrate conversion of strings to numbers for comparison operations:

```
mysql> SELECT 1 > '6x';
      -> 0
mysql> SELECT 7 > '6x';
      -> 1
mysql> SELECT 0 > 'x6';
      -> 0
mysql> SELECT 0 = 'x6';
      -> 1
```

Note that when you are comparing a string column with a number, MySQL cannot use an index on the column to look up the value quickly. If *str_col* is an indexed string column, the index cannot be used when performing the lookup in the following statement:

```
SELECT * FROM tbl_name WHERE str_col=1;
```

The reason for this is that there are many different strings that may convert to the value **1**, such as **'1'**, **' 1'**, or **'1a'**.

Comparisons that use floating-point numbers (or values that are converted to floating-point numbers) are approximate because such numbers are inexact. This might lead to results that appear inconsistent:

```
mysql> SELECT '18015376320243458' = 18015376320243458;
      -> 1
mysql> SELECT '18015376320243459' = 18015376320243459;
      -> 0
```

Such results can occur because the values are converted to floating-point numbers, which have only 53 bits of precision and are subject to rounding:

```
mysql> SELECT '18015376320243459'+0.0;
      -> 1.8015376320243e+16
```

Furthermore, the conversion from string to floating-point and from integer to floating-point do not necessarily occur the same way. The integer may be converted to floating-point by the CPU, whereas the string is converted digit by digit in an operation that involves floating-point multiplications.

The results shown will vary on different systems, and can be affected by factors such as computer architecture or the compiler version or optimization level. One way to avoid such problems is to use **CAST()** so that a value will not be converted implicitly to a float-point number:

```
mysql> SELECT CAST('18015376320243459' AS UNSIGNED) = 18015376320243459;
      -> 1
```

For more information about floating-point comparisons, see [Section B.1.5.8, “Problems with Floating-Point Comparisons”](#).

11.2.3. Comparison Functions and Operators

Name	Description
<code>BETWEEN ... AND ...</code>	Check whether a value is within a range of values
<code>COALESCE()</code>	Return the first non-NULL argument
<code><=></code>	NULL-safe equal to operator
<code>=</code>	Equal operator
<code>>=</code>	Greater than or equal operator
<code>></code>	Greater than operator
<code>GREATEST()</code>	Return the largest argument
<code>IN</code>	Check whether a value is within a set of values
<code>INTERVAL()</code>	Return the index of the argument that is less than the first argument
<code>IS NULL</code>	NULL value test
<code>IS</code>	Test a value against a boolean
<code>ISNULL()</code>	Test whether the argument is NULL
<code>LEAST()</code>	Return the smallest argument
<code><=</code>	Less than or equal operator
<code><</code>	Less than operator
<code>LIKE</code>	Simple pattern matching
<code>NOT BETWEEN ... AND ...</code>	Check whether a value is not within a range of values
<code>!=, <></code>	Not equal operator
<code>NOT IN</code>	Check whether a value is not within a set of values
<code>NOT LIKE</code>	Negation of simple pattern matching
<code>SOUNDS LIKE(v4.1.0)</code>	Compare sounds

Comparison operations result in a value of **1** (**TRUE**), **0** (**FALSE**), or **NULL**. These operations work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as necessary.

Some of the functions in this section return values other than **1** (**TRUE**), **0** (**FALSE**), or **NULL**. For example, `LEAST()` and `GREATEST()`. However, the value they return is based on comparison operations performed according to the rules described in [Section 11.2.2, “Type Conversion in Expression Evaluation”](#).

To convert a value to a specific type for comparison purposes, you can use the `CAST()` function. String values can be converted to a different character set using `CONVERT()`. See [Section 11.9, “Cast Functions and Operators”](#).

By default, string comparisons are not case sensitive and use the current character set. The default is `latin1` (cp1252 West European), which also works well for English.

- =**

Equal:

```
mysql> SELECT 1 = 0;
-> 0
mysql> SELECT '0' = 0;
-> 1
mysql> SELECT '0.0' = 0;
-> 1
mysql> SELECT '0.01' = 0;
-> 0
mysql> SELECT '.01' = 0.01;
-> 1
```

- <=>**

NULL-safe equal. This operator performs an equality comparison like the `=` operator, but returns **1** rather than **NULL** if both operands are **NULL**, and **0** rather than **NULL** if one operand is **NULL**.

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
-> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
-> 1, NULL, NULL
```

- `<>`, `!=`

Not equal:

```
mysql> SELECT '.01' <> '0.01';
-> 1
mysql> SELECT .01 <> '0.01';
-> 0
mysql> SELECT 'zapp' <> 'zappp';
-> 1
```

- `<=`

Less than or equal:

```
mysql> SELECT 0.1 <= 2;
-> 1
```

- `<`

Less than:

```
mysql> SELECT 2 < 2;
-> 0
```

- `>=`

Greater than or equal:

```
mysql> SELECT 2 >= 2;
-> 1
```

- `>`

Greater than:

```
mysql> SELECT 2 > 2;
-> 0
```

- `IS boolean_value`, `IS NOT boolean_value`

Tests a value against a boolean value, where *boolean_value* can be `TRUE`, `FALSE`, or `UNKNOWN`.

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
-> 1, 1, 1
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
-> 1, 1, 0
```

- `IS NULL`, `IS NOT NULL`

Tests whether a value is or is not `NULL`.

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
-> 0, 0, 1
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
-> 1, 1, 0
```

To work well with ODBC programs, MySQL supports the following extra features when using `IS NULL`:

- You can find the row that contains the most recent `AUTO_INCREMENT` value by issuing a statement of the following form immediately after generating the value:


```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

This behavior can be disabled by setting `SQL_AUTO_IS_NULL=0`. See [Section 12.5.3, “SET Syntax”](#).

- For `DATE` and `DATETIME` columns that are declared as `NOT NULL`, you can find the special date `'0000-00-00'` by using a statement like this:

```
SELECT * FROM tbl_name WHERE date_column IS NULL
```

This is needed to get some ODBC applications to work because ODBC does not support a `'0000-00-00'` date value.

- `expr BETWEEN min AND max`

If `expr` is greater than or equal to `min` and `expr` is less than or equal to `max`, `BETWEEN` returns `1`, otherwise it returns `0`. This is equivalent to the expression `(min <= expr AND expr <= max)` if all the arguments are of the same type. Otherwise type conversion takes place according to the rules described in [Section 11.2.2, “Type Conversion in Expression Evaluation”](#), but applied to all the three arguments.

```
mysql> SELECT 1 BETWEEN 2 AND 3;
-> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
-> 1
mysql> SELECT 2 BETWEEN 2 AND '3';
-> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3';
-> 0
```

For best results when using `BETWEEN` with date or time values, you should use `CAST()` to explicitly convert the values to the desired data type. Examples: If you compare a `DATETIME` to two `DATE` values, convert the `DATE` values to `DATETIME` values. If you use a string constant such as `'2001-1-1'` in a comparison to a `DATE`, cast the string to a `DATE`.

- `expr NOT BETWEEN min AND max`

This is the same as `NOT (expr BETWEEN min AND max)`.

- `COALESCE(value, ...)`

Returns the first non-`NULL` value in the list, or `NULL` if there are no non-`NULL` values.

```
mysql> SELECT COALESCE(NULL,1);
-> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
-> NULL
```

- `GREATEST(value1,value2,...)`

With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for `LEAST()`.

```
mysql> SELECT GREATEST(2,0);
-> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
-> 767.0
mysql> SELECT GREATEST('B','A','C');
-> 'C'
```

`GREATEST()` returns `NULL` if any argument is `NULL`.

- `expr IN (value,...)`

Returns `1` if `expr` is equal to any of the values in the `IN` list, else returns `0`. If all values are constants, they are evaluated according to the type of `expr` and sorted. The search for the item then is done using a binary search. This means `IN` is very quick if the `IN` value list consists entirely of constants. Otherwise, type conversion takes place according to the rules described in [Section 11.2.2, “Type Conversion in Expression Evaluation”](#), but applied to all the arguments.

```
mysql> SELECT 2 IN (0,3,5,7);
-> 0
```

```
mysql> SELECT 'wefwf' IN ('wee', 'wefwf', 'weg');
-> 1
```

You should never mix quoted and unquoted values in an **IN** list because the comparison rules for quoted values (such as strings) and unquoted values (such as numbers) differ. Mixing types may therefore lead to inconsistent results. For example, do not write an **IN** expression like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN (1,2,'a');
```

Instead, write it like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN ('1','2','a');
```

The number of values in the **IN** list is only limited by the `max_allowed_packet` value.

To comply with the SQL standard, **IN** returns **NULL** not only if the expression on the left hand side is **NULL**, but also if no match is found in the list and one of the expressions in the list is **NULL**.

IN() syntax can also be used to write certain types of subqueries. See [Section 12.2.8.3, “Subqueries with ANY, IN, and SOME”](#).

- **expr NOT IN (value, ...)**

This is the same as **NOT (expr IN (value, ...))**.

- **ISNULL(expr)**

If *expr* is **NULL**, **ISNULL()** returns **1**, otherwise it returns **0**.

```
mysql> SELECT ISNULL(1+1);
-> 0
mysql> SELECT ISNULL(1/0);
-> 1
```

ISNULL() can be used instead of **=** to test whether a value is **NULL**. (Comparing a value to **NULL** using **=** always yields false.)

The **ISNULL()** function shares some special behaviors with the **IS NULL** comparison operator. See the description of **IS NULL**.

- **INTERVAL(N, N1, N2, N3, ...)**

Returns **0** if $N < N1$, **1** if $N < N2$ and so on or **-1** if N is **NULL**. All arguments are treated as integers. It is required that $N1 < N2 < N3 < \dots < Nn$ for this function to work correctly. This is because a binary search is used (very fast).

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
-> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
-> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
-> 0
```

- **LEAST(value1, value2, ...)**

With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

- If the return value is used in an **INTEGER** context or all arguments are integer-valued, they are compared as integers.
- If the return value is used in a **REAL** context or all arguments are real-valued, they are compared as reals.
- If any argument is a case-sensitive string, the arguments are compared as case-sensitive strings.
- In all other cases, the arguments are compared as case-insensitive strings.

LEAST() returns **NULL** if any argument is **NULL**.

```
mysql> SELECT LEAST(2, 0);
-> 0
mysql> SELECT LEAST(34.0, 3.0, 5.0, 767.0);
-> 3.0
```

```
mysql> SELECT LEAST('B', 'A', 'C');
-> 'A'
```

Note that the preceding conversion rules can produce strange results in some borderline cases:

```
mysql> SELECT CAST(LEAST(3600, 9223372036854775808.0) as SIGNED);
-> -9223372036854775808
```

This happens because MySQL reads `9223372036854775808.0` in an integer context. The integer representation is not good enough to hold the value, so it wraps to a signed integer.

11.2.4. Logical Operators

Name	Description
AND, &&	Logical AND
NOT, !	Negates value
, OR	Logical OR
XOR	Logical XOR

In SQL, all logical operators evaluate to `TRUE`, `FALSE`, or `NULL` (`UNKNOWN`). In MySQL, these are implemented as `1` (`TRUE`), `0` (`FALSE`), and `NULL`. Most of this is common to different SQL database servers, although some servers may return any non-zero value for `TRUE`.

Note that MySQL evaluates any non-zero or non-`NULL` value to `TRUE`. For example, the following statements all assess to `TRUE`:

```
mysql> SELECT 10 IS TRUE;
-> 1
mysql> SELECT -10 IS TRUE;
-> 1
mysql> SELECT 'string' IS NOT NULL;
-> 1
```

- **NOT, !**

Logical NOT. Evaluates to `1` if the operand is `0`, to `0` if the operand is non-zero, and `NOT NULL` returns `NULL`.

```
mysql> SELECT NOT 10;
-> 0
mysql> SELECT NOT 0;
-> 1
mysql> SELECT NOT NULL;
-> NULL
mysql> SELECT ! (1+1);
-> 0
mysql> SELECT ! 1+1;
-> 1
```

The last example produces `1` because the expression evaluates the same way as `(!1)+1`.

- **AND, &&**

Logical AND. Evaluates to `1` if all operands are non-zero and not `NULL`, to `0` if one or more operands are `0`, otherwise `NULL` is returned.

```
mysql> SELECT 1 && 1;
-> 1
mysql> SELECT 1 && 0;
-> 0
mysql> SELECT 1 && NULL;
-> NULL
mysql> SELECT 0 && NULL;
-> 0
mysql> SELECT NULL && 0;
-> 0
```

- **OR, ||**

Logical OR. When both operands are non-**NULL**, the result is **1** if any operand is non-zero, and **0** otherwise. With a **NULL** operand, the result is **1** if the other operand is non-zero, and **NULL** otherwise. If both operands are **NULL**, the result is **NULL**.

```
mysql> SELECT 1 || 1;
-> 1
mysql> SELECT 1 || 0;
-> 1
mysql> SELECT 0 || 0;
-> 0
mysql> SELECT 0 || NULL;
-> NULL
mysql> SELECT 1 || NULL;
-> 1
```

- **XOR**

Logical XOR. Returns **NULL** if either operand is **NULL**. For non-**NULL** operands, evaluates to **1** if an odd number of operands is non-zero, otherwise **0** is returned.

```
mysql> SELECT 1 XOR 1;
-> 0
mysql> SELECT 1 XOR 0;
-> 1
mysql> SELECT 1 XOR NULL;
-> NULL
mysql> SELECT 1 XOR 1 XOR 1;
-> 1
```

$a \text{ XOR } b$ is mathematically equal to $(a \text{ AND } (\text{NOT } b)) \text{ OR } ((\text{NOT } a) \text{ and } b)$.

11.3. Control Flow Functions

Name	Description
CASE	Case statement
IF()	If/else construct
IFNULL()	Null if/else construct
NULLIF()	Return NULL if expr1 = expr2

- **CASE** *value* **WHEN** [*compare_value*] **THEN** *result* [**WHEN** [*compare_value*] **THEN** *result* ...] [**ELSE** *result*] **END**

CASE **WHEN** [*condition*] **THEN** *result* [**WHEN** [*condition*] **THEN** *result* ...] [**ELSE** *result*] **END**

The first version returns the *result* where *value=compare_value*. The second version returns the result for the first condition that is true. If there was no matching result value, the result after **ELSE** is returned, or **NULL** if there is no **ELSE** part.

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
-> WHEN 2 THEN 'two' ELSE 'more' END;
-> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
-> 'true'
mysql> SELECT CASE BINARY 'B'
-> WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
-> NULL
```

The default return type of a **CASE** expression is the compatible aggregated type of all return values, but also depends on the context in which it is used. If used in a string context, the result is returned as a string. If used in a numeric context, then the result is returned as a decimal, real, or integer value.

Note

The syntax of the **CASE** expression shown here differs slightly from that of the SQL **CASE** statement described in [Section 19.2.10.2, “CASE Statement”](#), for use inside stored routines. The **CASE** statement cannot have an **ELSE NULL**

■ clause, and it is terminated with **END CASE** instead of **END**.

- **IF(*expr1*,*expr2*,*expr3*)**

If *expr1* is **TRUE** (*expr1* <> 0 and *expr1* <> NULL) then **IF()** returns *expr2*; otherwise it returns *expr3*. **IF()** returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IF(1>2,2,3);
-> 3
mysql> SELECT IF(1<2, 'yes', 'no');
-> 'yes'
mysql> SELECT IF(STRCMP('test', 'test1'), 'no', 'yes');
-> 'no'
```

If only one of *expr2* or *expr3* is explicitly **NULL**, the result type of the **IF()** function is the type of the non-**NULL** expression.

expr1 is evaluated as an integer value, which means that if you are testing floating-point or string values, you should do so using a comparison operation.

```
mysql> SELECT IF(0.1,1,0);
-> 0
mysql> SELECT IF(0.1<>0,1,0);
-> 1
```

In the first case shown, **IF(0.1)** returns 0 because 0.1 is converted to an integer value, resulting in a test of **IF(0)**. This may not be what you expect. In the second case, the comparison tests the original floating-point value to see whether it is non-zero. The result of the comparison is used as an integer.

The default return type of **IF()** (which may matter when it is stored into a temporary table) is calculated as follows:

Expression	Return Value
<i>expr2</i> or <i>expr3</i> returns a string	string
<i>expr2</i> or <i>expr3</i> returns a floating-point value	floating-point
<i>expr2</i> or <i>expr3</i> returns an integer	integer

If *expr2* and *expr3* are both strings, the result is case sensitive if either string is case sensitive.

Note

There is also an **IF statement**, which differs from the **IF()** function described here. See [Section 19.2.10.1, “IF Statement”](#).

- **IFNULL(*expr1*,*expr2*)**

If *expr1* is not **NULL**, **IFNULL()** returns *expr1*; otherwise it returns *expr2*. **IFNULL()** returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IFNULL(1,0);
-> 1
mysql> SELECT IFNULL(NULL,10);
-> 10
mysql> SELECT IFNULL(1/0,10);
-> 10
mysql> SELECT IFNULL(1/0, 'yes');
-> 'yes'
```

The default result value of **IFNULL(*expr1*,*expr2*)** is the more “general” of the two expressions, in the order **STRING**, **REAL**, or **INTEGER**. Consider the case of a table based on expressions or where MySQL must internally store a value returned by **IFNULL()** in a temporary table:

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| test  | char(4) |      |      |          |       |
+-----+-----+-----+-----+-----+
```

In this example, the type of the `test` column is `CHAR(4)`.

- `NULLIF(expr1,expr2)`

Returns `NULL` if `expr1 = expr2` is true, otherwise returns `expr1`. This is the same as `CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END`.

```
mysql> SELECT NULLIF(1,1);
-> NULL
mysql> SELECT NULLIF(1,2);
-> 1
```

Note that MySQL evaluates `expr1` twice if the arguments are not equal.

11.4. String Functions

Name	Description
<code>ASCII()</code>	Return numeric value of left-most character
<code>BIN()</code>	Return a string representation of the argument
<code>BIT_LENGTH()</code>	Return length of argument in bits
<code>CHAR_LENGTH()</code>	Return number of characters in argument
<code>CHAR()</code>	Return the character for each integer passed
<code>CHARACTER_LENGTH()</code>	A synonym for <code>CHAR_LENGTH()</code>
<code>CONCAT_WS()</code>	Return concatenate with separator
<code>CONCAT()</code>	Return concatenated string
<code>CONV()</code>	Convert numbers between different number bases
<code>ELT()</code>	Return string at index number
<code><=></code>	NULL-safe equal to operator
<code>=</code>	Equal operator
<code>EXPORT_SET()</code>	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
<code>FIELD()</code>	Return the index (position) of the first argument in the subsequent arguments
<code>FIND_IN_SET()</code>	Return the index position of the first argument within the second argument
<code>FORMAT()</code>	Return a number formatted to specified number of decimal places
<code>>=</code>	Greater than or equal operator
<code>></code>	Greater than operator
<code>HEX()</code>	Return a string representation of a hex value
<code>INSERT()</code>	Insert a substring at the specified position up to the specified number of characters
<code>INSTR()</code>	Return the index of the first occurrence of substring
<code>IS NULL</code>	NULL value test
<code>IS</code>	Test a value against a boolean
<code>LCASE()</code>	Synonym for <code>LOWER()</code>
<code>LEFT()</code>	Return the leftmost number of characters as specified
<code>LENGTH()</code>	Return the length of a string in bytes
<code><=</code>	Less than or equal operator
<code><</code>	Less than operator
<code>LIKE</code>	Simple pattern matching
<code>LOAD_FILE()</code>	Load the named file

Name	Description
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MID()	Return a substring starting from the specified position
!=, <>	Not equal operator
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OCT()	Return a string representation of the octal argument
OCTET_LENGTH()	A synonym for LENGTH()
ORD()	If the leftmost character of the argument is a multi-byte character, returns the code for that character
POSITION()	A synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Pattern matching using regular expressions
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Synonym for REGEXP
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SOUNDEX()	Return a soundex string
SOUNDS LIKE(v4.1.0)	Compare sounds
SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING(), SUBSTR()	Return the substring as specified
TRIM()	Remove leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()(v4.1.2)	Convert each pair of hexadecimal digits to a character
UPPER()	Convert to uppercase

String-valued functions return **NULL** if the length of the result would be greater than the value of the `max_allowed_packet` system variable. See [Section 6.5.2, “Tuning Server Parameters”](#).

For functions that operate on string positions, the first position is numbered 1.

For functions that take length arguments, non-integer arguments are rounded to the nearest integer.

- [ASCII\(str\)](#)

Returns the numeric value of the leftmost character of the string *str*. Returns 0 if *str* is the empty string. Returns **NULL** if *str* is **NULL**. [ASCII\(\)](#) works for 8-bit characters.

```
mysql> SELECT ASCII('2');
-> 50
mysql> SELECT ASCII(2);
-> 50
mysql> SELECT ASCII('dx');
-> 100
```

See also the `ORD()` function.

- `BIN(N)`

Returns a string representation of the binary value of *N*, where *N* is a longlong (`BIGINT`) number. This is equivalent to `CONV(N, 10, 2)`. Returns `NULL` if *N* is `NULL`.

```
mysql> SELECT BIN(12);
-> '1100'
```

- `BIT_LENGTH(str)`

Returns the length of the string *str* in bits.

```
mysql> SELECT BIT_LENGTH('text');
-> 32
```

- `CHAR(N, ... [USING charset_name])`

`CHAR()` interprets each argument *N* as an integer and returns a string consisting of the characters given by the code values of those integers. `NULL` values are skipped.

```
mysql> SELECT CHAR(77,121,83,81,'76');
-> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
-> 'MMM'
```

`CHAR()` arguments larger than 255 are converted into multiple result bytes. For example, `CHAR(256)` is equivalent to `CHAR(1, 0)`, and `CHAR(256*256)` is equivalent to `CHAR(1, 0, 0)`:

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+-----+-----+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+-----+-----+
| 0100          | 0100          |
+-----+-----+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+-----+-----+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+-----+-----+
| 010000         | 010000         |
+-----+-----+
```

By default, `CHAR()` returns a binary string. To produce a string in a given character set, use the optional `USING` clause:

```
mysql> SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8));
+-----+-----+
| CHARSET(CHAR(0x65)) | CHARSET(CHAR(0x65 USING utf8)) |
+-----+-----+
| binary              | utf8                          |
+-----+-----+
```

If `USING` is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from `CHAR()` becomes `NULL`.

- `CHAR_LENGTH(str)`

Returns the length of the string *str*, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five two-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

- `CHARACTER_LENGTH(str)`

`CHARACTER_LENGTH()` is a synonym for `CHAR_LENGTH()`.

- `CONCAT(str1, str2, ...)`

Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are non-binary strings, the result is a non-binary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form; if you want to avoid that, you can use an explicit type cast, as in this example:

```
SELECT CONCAT(CAST(int_col AS CHAR), char_col);
```

`CONCAT()` returns `NULL` if any argument is `NULL`.

```
mysql> SELECT CONCAT('My', 'S', 'QL');
-> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
-> NULL
mysql> SELECT CONCAT(14.3);
-> '14.3'
```

- `CONCAT_WS(separator, str1, str2, ...)`

`CONCAT_WS()` stands for Concatenate With Separator and is a special form of `CONCAT()`. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is `NULL`, the result is `NULL`.

```
mysql> SELECT CONCAT_WS(',', 'First name', 'Second name', 'Last Name');
-> 'First name,Second name,Last Name'
mysql> SELECT CONCAT_WS(',', 'First name', NULL, 'Last Name');
-> 'First name,Last Name'
```

`CONCAT_WS()` does not skip empty strings. However, it does skip any `NULL` values after the separator argument.

- `CONV(N, from_base, to_base)`

Converts numbers between different number bases. Returns a string representation of the number `N`, converted from base `from_base` to base `to_base`. Returns `NULL` if any argument is `NULL`. The argument `N` is interpreted as an integer, but may be specified as an integer or a string. The minimum base is `2` and the maximum base is `36`. If `to_base` is a negative number, `N` is regarded as a signed number. Otherwise, `N` is treated as unsigned. `CONV()` works with 64-bit precision.

```
mysql> SELECT CONV('a',16,2);
-> '1010'
mysql> SELECT CONV('6E',18,8);
-> '172'
mysql> SELECT CONV(-17,10,-18);
-> '-H'
mysql> SELECT CONV(10+'10'+ '10'+0xa,10,10);
-> '40'
```

- `ELT(N, str1, str2, str3, ...)`

Returns `str1` if `N = 1`, `str2` if `N = 2`, and so on. Returns `NULL` if `N` is less than `1` or greater than the number of arguments. `ELT()` is the complement of `FIELD()`.

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
-> 'ej'
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
-> 'foo'
```

- `EXPORT_SET(bits, on, off[, separator[, number_of_bits]])`

Returns a string such that for every bit set in the value `bits`, you get an `on` string and for every bit not set in the value, you get an `off` string. Bits in `bits` are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the `separator` string (the default being the comma character `,`). The number of bits examined is given by `number_of_bits` (defaults to 64).

```
mysql> SELECT EXPORT_SET(5, 'Y', 'N', ',', 4);
-> 'Y,N,Y,N'
mysql> SELECT EXPORT_SET(6, '1', '0', ',', 10);
-> '0,1,1,0,0,0,0,0,0,0'
```

- **FIELD(*str*, *str1*, *str2*, *str3*, ...)**

Returns the index (position) of *str* in the *str1*, *str2*, *str3*, ... list. Returns 0 if *str* is not found.

If all arguments to **FIELD()** are strings, all arguments are compared as strings. If all arguments are numbers, they are compared as numbers. Otherwise, the arguments are compared as double.

If *str* is **NULL**, the return value is 0 because **NULL** fails equality comparison with any value. **FIELD()** is the complement of **ELT()**.

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 0
```

- **FIND_IN_SET(*str*, *strlist*)**

Returns a value in the range of 1 to *N* if the string *str* is in the string list *strlist* consisting of *N* substrings. A string list is a string composed of substrings separated by ',' characters. If the first argument is a constant string and the second is a column of type **SET**, the **FIND_IN_SET()** function is optimized to use bit arithmetic. Returns 0 if *str* is not in *strlist* or if *strlist* is the empty string. Returns **NULL** if either argument is **NULL**. This function does not work properly if the first argument contains a comma (',') character.

```
mysql> SELECT FIND_IN_SET('b', 'a,b,c,d');
-> 2
```

- **FORMAT(*X*, *D*)**

Formats the number *X* to a format like '#,###,###.###', rounded to *D* decimal places, and returns the result as a string. If *D* is 0, the result has no decimal point or fractional part.

```
mysql> SELECT FORMAT(12332.123456, 4);
-> '12,332.1235'
mysql> SELECT FORMAT(12332.1, 4);
-> '12,332.1000'
mysql> SELECT FORMAT(12332.2, 0);
-> '12,332'
```

- **HEX(*N_or_S*)**

If *N_or_S* is a number, returns a string representation of the hexadecimal value of *N*, where *N* is a longlong (**BIGINT**) number. This is equivalent to **CONV(*N*, 10, 16)**.

If *N_or_S* is a string, returns a hexadecimal string representation of *N_or_S* where each character in *N_or_S* is converted to two hexadecimal digits.

```
mysql> SELECT HEX(255);
-> 'FF'
mysql> SELECT HEX(0x616263);
-> 'abc'
mysql> SELECT HEX('abc');
-> 616263
```

- **INSERT(*str*, *pos*, *len*, *newstr*)**

Returns the string *str*, with the substring beginning at position *pos* and *len* characters long replaced by the string *newstr*. Returns the original string if *pos* is not within the length of the string. Replaces the rest of the string from position *pos* if *len* is not within the length of the rest of the string. Returns **NULL** if any argument is **NULL**.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
-> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
-> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
-> 'QuWhat'
```

This function is multi-byte safe.

- `INSTR(str, substr)`

Returns the position of the first occurrence of substring `substr` in string `str`. This is the same as the two-argument form of `LOCATE()`, except that the order of the arguments is reversed.

```
mysql> SELECT INSTR('foobarbar', 'bar');
-> 4
mysql> SELECT INSTR('xbar', 'foobar');
-> 0
```

This function is multi-byte safe, and is case sensitive only if at least one argument is a binary string.

- `LCASE(str)`

`LCASE()` is a synonym for `LOWER()`.

- `LEFT(str, len)`

Returns the leftmost `len` characters from the string `str`, or `NULL` if any argument is `NULL`.

```
mysql> SELECT LEFT('foobarbar', 5);
-> 'fooba'
```

- `LENGTH(str)`

Returns the length of the string `str`, measured in bytes. A multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

```
mysql> SELECT LENGTH('text');
-> 4
```

- `LOAD_FILE(file_name)`

Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full pathname to the file, and you must have the `FILE` privilege. The file must be readable by all and its size less than `max_allowed_packet` bytes.

If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

As of MySQL 5.1.6, the `character_set_filesystem` system variable controls interpretation of filenames that are given as literal strings.

```
mysql> UPDATE t
      SET blob_col=LOAD_FILE('/tmp/picture')
      WHERE id=1;
```

- `LOCATE(substr, str)`, `LOCATE(substr, str, pos)`

The first syntax returns the position of the first occurrence of substring `substr` in string `str`. The second syntax returns the position of the first occurrence of substring `substr` in string `str`, starting at position `pos`. Returns `0` if `substr` is not in `str`.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
-> 4
mysql> SELECT LOCATE('xbar', 'foobar');
-> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
-> 7
```

This function is multi-byte safe, and is case-sensitive only if at least one argument is a binary string.

- `LOWER(str)`

Returns the string `str` with all characters changed to lowercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

```
mysql> SELECT LOWER('QUADRATICALLY');
-> 'quadratically'
```

This function is multi-byte safe.

- **LPAD(*str*, *len*, *padstr*)**

Returns the string *str*, left-padded with the string *padstr* to a length of *len* characters. If *str* is longer than *len*, the return value is shortened to *len* characters.

```
mysql> SELECT LPAD('hi',4,'?');
-> '??hi'
mysql> SELECT LPAD('hi',1,'?');
-> 'h'
```

- **LTRIM(*str*)**

Returns the string *str* with leading space characters removed.

```
mysql> SELECT LTRIM('  barbar');
-> 'barbar'
```

This function is multi-byte safe.

- **MAKE_SET(*bits*, *str1*, *str2*, ...)**

Returns a set value (a string containing substrings separated by ',' characters) consisting of the strings that have the corresponding bit in *bits* set. *str1* corresponds to bit 0, *str2* to bit 1, and so on. **NULL** values in *str1*, *str2*, ... are not appended to the result.

```
mysql> SELECT MAKE_SET(1,'a','b','c');
-> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
-> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
-> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
-> ''
```

- **MID(*str*, *pos*, *len*)**

MID(*str*, *pos*, *len*) is a synonym for **SUBSTRING(*str*, *pos*, *len*)**.

- **OCT(*N*)**

Returns a string representation of the octal value of *N*, where *N* is a longlong (**BIGINT**) number. This is equivalent to **CONV(*N*, 10, 8)**. Returns **NULL** if *N* is **NULL**.

```
mysql> SELECT OCT(12);
-> '14'
```

- **OCTET_LENGTH(*str*)**

OCTET_LENGTH() is a synonym for **LENGTH()**.

- **ORD(*str*)**

If the leftmost character of the string *str* is a multi-byte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
(1st byte code)
+ (2nd byte code × 256)
+ (3rd byte code × 2562) ...
```

If the leftmost character is not a multi-byte character, **ORD()** returns the same value as the **ASCII()** function.

```
mysql> SELECT ORD('2');
-> 50
```

- **POSITION(*substr* IN *str*)**

`POSITION(substr IN str)` is a synonym for `LOCATE(substr, str)`.

- `QUOTE(str)`

Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotes and with each instance of single quote (''), backslash ('\'), ASCII `NUL`, and Control-Z preceded by a backslash. If the argument is `NULL`, the return value is the word "NULL" without enclosing single quotes.

```
mysql> SELECT QUOTE('Don\'t!');
-> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
-> NULL
```

- `REPEAT(str, count)`

Returns a string consisting of the string `str` repeated `count` times. If `count` is less than 1, returns an empty string. Returns `NULL` if `str` or `count` are `NULL`.

```
mysql> SELECT REPEAT('MySQL', 3);
-> 'MySQLMySQLMySQL'
```

- `REPLACE(str, from_str, to_str)`

Returns the string `str` with all occurrences of the string `from_str` replaced by the string `to_str`. `REPLACE()` performs a case-sensitive match when searching for `from_str`.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'
```

This function is multi-byte safe.

- `REVERSE(str)`

Returns the string `str` with the order of the characters reversed.

```
mysql> SELECT REVERSE('abc');
-> 'cba'
```

This function is multi-byte safe.

- `RIGHT(str, len)`

Returns the rightmost `len` characters from the string `str`, or `NULL` if any argument is `NULL`.

```
mysql> SELECT RIGHT('foobarbar', 4);
-> 'rbar'
```

This function is multi-byte safe.

- `RPAD(str, len, padstr)`

Returns the string `str`, right-padded with the string `padstr` to a length of `len` characters. If `str` is longer than `len`, the return value is shortened to `len` characters.

```
mysql> SELECT RPAD('hi', 5, '?');
-> 'hi???'
mysql> SELECT RPAD('hi', 1, '?');
-> 'h'
```

This function is multi-byte safe.

- `RTRIM(str)`

Returns the string `str` with trailing space characters removed.

```
mysql> SELECT RTRIM('barbar ');
```

```
-> 'barbar'
```

This function is multi-byte safe.

- **SOUNDEX(*str*)**

Returns a soundex string from *str*. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the **SOUNDEX()** function returns an arbitrarily long string. You can use **SUBSTRING()** on the result to get a standard soundex string. All non-alphabetic characters in *str* are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

Important

When using **SOUNDEX()**, you should be aware of the following limitations:

- This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reliable results.
- This function is not guaranteed to provide consistent results with strings that use multi-byte character sets, including **utf-8**.

We hope to remove these limitations in a future release. See [Bug#22638](#) for more information.

```
mysql> SELECT SOUNDEX('Hello');
-> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
-> 'Q36324'
```

Note

This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

- **expr1 SOUNDS LIKE expr2**

This is the same as **SOUNDEX(expr1) = SOUNDEX(expr2)**.

- **SPACE(*N*)**

Returns a string consisting of *N* space characters.

```
mysql> SELECT SPACE(6);
-> '      '
```

- **SUBSTRING(*str*,*pos*), SUBSTRING(*str* FROM *pos*), SUBSTRING(*str*,*pos*,*len*), SUBSTRING(*str* FROM *pos* FOR *len*)**

The forms without a *len* argument return a substring from string *str* starting at position *pos*. The forms with a *len* argument return a substring *len* characters long from string *str*, starting at position *pos*. The forms that use **FROM** are standard SQL syntax. It is also possible to use a negative value for *pos*. In this case, the beginning of the substring is *pos* characters from the end of the string, rather than the beginning. A negative value may be used for *pos* in any of the forms of this function.

For all forms of **SUBSTRING()**, the position of the first character in the string from which the substring is to be extracted is reckoned as **1**.

```
mysql> SELECT SUBSTRING('Quadratically',5);
-> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
-> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
-> 'ratica'
mysql> SELECT SUBSTRING('Sakila', -3);
-> 'ila'
mysql> SELECT SUBSTRING('Sakila', -5, 3);
-> 'aki'
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
-> 'ki'
```

This function is multi-byte safe.

If *len* is less than 1, the result is the empty string.

SUBSTR() is a synonym for **SUBSTRING()**.

- **SUBSTRING_INDEX(*str*,*delim*,*count*)**

Returns the substring from string *str* before *count* occurrences of the delimiter *delim*. If *count* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *count* is negative, everything to the right of the final delimiter (counting from the right) is returned. **SUBSTRING_INDEX()** performs a case-sensitive match when searching for *delim*.

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
-> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
-> 'mysql.com'
```

This function is multi-byte safe.

- **TRIM([{BOTH | LEADING | TRAILING} [*remstr*] FROM] *str*), TRIM([*remstr* FROM] *str*)**

Returns the string *str* with all *remstr* prefixes or suffixes removed. If none of the specifiers **BOTH**, **LEADING**, or **TRAILING** is given, **BOTH** is assumed. *remstr* is optional and, if not specified, spaces are removed.

```
mysql> SELECT TRIM(' bar ');
-> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
-> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
-> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxyz');
-> 'barx'
```

This function is multi-byte safe.

- **UCASE(*str*)**

UCASE() is a synonym for **UPPER()**.

- **UNHEX(*str*)**

Performs the inverse operation of **HEX(*str*)**. That is, it interprets each pair of hexadecimal digits in the argument as a number and converts it to the character represented by the number. The resulting characters are returned as a binary string.

```
mysql> SELECT UNHEX('4D7953514C');
-> 'MySQL'
mysql> SELECT 0x4D7953514C;
-> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
-> 'string'
mysql> SELECT HEX(UNHEX('1267'));
-> '1267'
```

The characters in the argument string must be legal hexadecimal digits: '0' .. '9', 'A' .. 'F', 'a' .. 'f'. If **UNHEX()** encounters any non-hexadecimal digits in the argument, it returns **NULL**:

```
mysql> SELECT UNHEX('GG');
+-----+
| UNHEX('GG') |
+-----+
| NULL        |
+-----+
```

A **NULL** result can occur if the argument to **UNHEX()** is a **BINARY** column, because values are padded with 0x00 bytes when stored but those bytes are not stripped on retrieval. For example 'aa' is stored into a **CHAR(3)** column as 'aa ' and retrieved as 'aa' (with the trailing pad space stripped), so **UNHEX()** for the column value returns 'A'. By contrast 'aa' is stored into a **BINARY(3)** column as 'aa\0' and retrieved as 'aa\0' (with the trailing pad 0x00 byte not stripped). '\0' is not a legal hexadecimal digit, so **UNHEX()** for the column value returns **NULL**.

- `UPPER(str)`

Returns the string *str* with all characters changed to uppercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

```
mysql> SELECT UPPER('Hej');
-> 'HEJ'
```

This function is multi-byte safe.

11.4.1. String Comparison Functions

Name	Description
<code><=></code>	NULL-safe equal to operator
<code>=</code>	Equal operator
<code>>=</code>	Greater than or equal operator
<code>></code>	Greater than operator
<code>IS NULL</code>	NULL value test
<code>IS</code>	Test a value against a boolean
<code><=</code>	Less than or equal operator
<code><</code>	Less than operator
<code>LIKE</code>	Simple pattern matching
<code>!=, <></code>	Not equal operator
<code>NOT LIKE</code>	Negation of simple pattern matching
<code>SOUNDS LIKE</code> (v4.1.0)	Compare sounds

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This affects only comparisons.

Normally, if any expression in a string comparison is case sensitive, the comparison is performed in case-sensitive fashion.

- `expr LIKE pat [ESCAPE 'escape_char']`

Pattern matching using SQL simple regular expression comparison. Returns `1` (TRUE) or `0` (FALSE). If either *expr* or *pat* is `NULL`, the result is `NULL`.

The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

Per the SQL standard, `LIKE` performs matching on a per-character basis, thus it can produce results different from the `=` comparison operator:

```
mysql> SELECT 'ä' LIKE 'ae' COLLATE latin1_german2_ci;
+-----+
| 'ä' LIKE 'ae' COLLATE latin1_german2_ci |
+-----+
|                                         0 |
+-----+
mysql> SELECT 'ä' = 'ae' COLLATE latin1_german2_ci;
+-----+
| 'ä' = 'ae' COLLATE latin1_german2_ci |
+-----+
|                                         1 |
+-----+
```

With `LIKE` you can use the following two wildcard characters in the pattern:

Character	Description
<code>%</code>	Matches any number of characters, even zero characters

<code>_</code>	Matches exactly one character

```
mysql> SELECT 'David!' LIKE 'David_';
-> 1
mysql> SELECT 'David!' LIKE '%D%v%';
-> 1
```

To test for literal instances of a wildcard character, precede it by the escape character. If you do not specify the `ESCAPE` character, `'\'` is assumed.

String	Description
<code>\%</code>	Matches one <code>'%'</code> character
<code>_</code>	Matches one <code>'_'</code> character

```
mysql> SELECT 'David!' LIKE 'David\_';
-> 0
mysql> SELECT 'David_' LIKE 'David\_';
-> 1
```

To specify a different escape character, use the `ESCAPE` clause:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
-> 1
```

The escape sequence should be empty or one character long. As of MySQL 5.1.2, if the `NO_BACKSLASH_ESCAPES` SQL mode is enabled, the sequence cannot be empty.

The following two statements illustrate that string comparisons are not case sensitive unless one of the operands is a binary string:

```
mysql> SELECT 'abc' LIKE 'ABC';
-> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
-> 0
```

In MySQL, `LIKE` is allowed on numeric expressions. (This is an extension to the standard SQL `LIKE`.)

```
mysql> SELECT 10 LIKE '1%';
-> 1
```

Note

Because MySQL uses C escape syntax in strings (for example, `'\n'` to represent a newline character), you must double any `'\'` that you use in `LIKE` strings. For example, to search for `'\n'`, specify it as `'\\n'`. To search for `'\'`, specify it as `'\\\\'`; this is because the backslashes are stripped once by the parser and again when the pattern match is made, leaving a single backslash to be matched against. (Exception: At the end of the pattern string, backslash can be specified as `'\\'`. At the end of the string, backslash stands for itself because there is nothing following to escape.)

- `expr NOT LIKE pat [ESCAPE 'escape_char']`

This is the same as `NOT (expr LIKE pat [ESCAPE 'escape_char'])`.

Note

Aggregate queries involving `NOT LIKE` comparisons with columns containing `NULL` may yield unexpected results. For example, consider the following table and data:

```
CREATE TABLE foo (bar VARCHAR(10));
INSERT INTO foo VALUES (NULL), (NULL);
```

The query `SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%';` returns 0. You might assume that `SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%';` would return 2. However, this is not the case: The second query returns 0. This is because `NULL NOT LIKE expr` always returns `NULL`, regardless of the value of

expr. The same is true for aggregate queries involving **NULL** and comparisons using **NOT RLIKE** or **NOT REGEXP**. In such cases, you must test explicitly for **NOT NULL** using **OR** (and not **AND**), as shown here:

```
SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%' OR bar IS NULL;
```

- **expr NOT REGEXP pat, expr NOT RLIKE pat**

This is the same as **NOT (expr REGEXP pat)**.

- **expr REGEXP pat, expr RLIKE pat**

Performs a pattern match of a string expression *expr* against a pattern *pat*. The pattern can be an extended regular expression. The syntax for regular expressions is discussed in [Section 11.4.2, “Regular Expressions”](#). Returns **1** if *expr* matches *pat*; otherwise it returns **0**. If either *expr* or *pat* is **NULL**, the result is **NULL**. **RLIKE** is a synonym for **REGEXP**, provided for **MySQL** compatibility.

The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

Note

Because MySQL uses the C escape syntax in strings (for example, `'\n'` to represent the newline character), you must double any `'\'` that you use in your **REGEXP** strings.

REGEXP is not case sensitive, except when used with binary strings.

```
mysql> SELECT 'Monty!' REGEXP 'm%y%';
-> 0
mysql> SELECT 'Monty!' REGEXP '.*';
-> 1
mysql> SELECT 'new*\n*line' REGEXP 'new\\*\\.\\*line';
-> 1
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
-> 1 0
mysql> SELECT 'a' REGEXP '^[a-d]';
-> 1
```

REGEXP and **RLIKE** use the current character set when deciding the type of a character. The default is **latin1** (cp1252 West European).

Warning

The **REGEXP** and **RLIKE** operators work in byte-wise fashion, so they are not multi-byte safe and may produce unexpected results with multi-byte character sets.

- **STRCMP(expr1, expr2)**

STRCMP() returns **0** if the strings are the same, **-1** if the first argument is smaller than the second according to the current sort order, and **1** otherwise.

```
mysql> SELECT STRCMP('text', 'text2');
-> -1
mysql> SELECT STRCMP('text2', 'text');
-> 1
mysql> SELECT STRCMP('text', 'text');
-> 0
```

STRCMP() uses the current character set when performing comparisons. This makes the default comparison behavior case insensitive unless one or both of the operands are binary strings.

11.4.2. Regular Expressions

Name	Description
NOT REGEXP	Negation of REGEXP
REGEXP	Pattern matching using regular expressions
RLIKE	Synonym for REGEXP

A regular expression is a powerful way of specifying a pattern for a complex search.

MySQL uses Henry Spencer's implementation of regular expressions, which is aimed at conformance with POSIX 1003.2. See [Appendix E, Credits](#). MySQL uses the extended version to support pattern-matching operations performed with the `REGEXP` operator in SQL statements. See [Section 3.3.4.7, “Pattern Matching”](#), and [Section 11.4.1, “String Comparison Functions”](#).

This section is a summary, with examples, of the special characters and constructs that can be used in MySQL for `REGEXP` operations. It does not contain all the details that can be found in Henry Spencer's `regex(7)` manual page. That manual page is included in MySQL source distributions, in the `regex.7` file under the `regex` directory.

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression `hello` matches `hello` and nothing else.

Non-trivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression `hello|word` matches either the string `hello` or the string `word`.

As a more complex example, the regular expression `B[an]*s` matches any of the strings `Bananas`, `Baaaaaas`, `Bs`, and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

A regular expression for the `REGEXP` operator may use any of the following special characters and constructs:

- `^`

Match the beginning of a string.

```
mysql> SELECT 'fo\info' REGEXP '^fo$';          -> 0
mysql> SELECT 'fofo' REGEXP '^fo';              -> 1
```

- `$`

Match the end of a string.

```
mysql> SELECT 'fo\no' REGEXP '^fo\no$';         -> 1
mysql> SELECT 'fo\no' REGEXP '^fo$';            -> 0
```

- `.`

Match any character (including carriage return and newline).

```
mysql> SELECT 'fofo' REGEXP '^f.*$';             -> 1
mysql> SELECT 'fo\r\info' REGEXP '^f.*$';       -> 1
```

- `a*`

Match any sequence of zero or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';              -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';           -> 1
mysql> SELECT 'Bn' REGEXP '^Ba*n';              -> 1
```

- `a+`

Match any sequence of one or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';              -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';              -> 0
```

- `a?`

Match either zero or one `a` character.

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';              -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';             -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';            -> 0
```

- `de|abc`

Match either of the sequences `de` or `abc`.

```
mysql> SELECT 'pi' REGEXP 'pi|apa';      -> 1
mysql> SELECT 'axe' REGEXP 'pi|apa';     -> 0
mysql> SELECT 'apa' REGEXP 'pi|apa';     -> 1
mysql> SELECT 'apa' REGEXP '^ (pi|apa)$'; -> 1
mysql> SELECT 'pi' REGEXP '^ (pi|apa)$';  -> 1
mysql> SELECT 'pix' REGEXP '^ (pi|apa)$'; -> 0
```

- `(abc)*`

Match zero or more instances of the sequence `abc`.

```
mysql> SELECT 'pi' REGEXP '^ (pi)*$';    -> 1
mysql> SELECT 'pip' REGEXP '^ (pi)*$';   -> 0
mysql> SELECT 'pipi' REGEXP '^ (pi)*$';  -> 1
```

- `{1}`, `{2, 3}`

`{n}` or `{m, n}` notation provides a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. `m` and `n` are integers.

- `a*`

Can be written as `a{0, }`.

- `a+`

Can be written as `a{1, }`.

- `a?`

Can be written as `a{0, 1}`.

To be more precise, `a{n}` matches exactly `n` instances of `a`. `a{n, }` matches `n` or more instances of `a`. `a{m, n}` matches `m` through `n` instances of `a`, inclusive.

`m` and `n` must be in the range from 0 to `RE_DUP_MAX` (default 255), inclusive. If both `m` and `n` are given, `m` must be less than or equal to `n`.

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e'; -> 0
mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e'; -> 1
mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e'; -> 1
```

- `[a-dX]`, `[^a-dX]`

Matches any character that is (or is not, if `^` is used) either `a`, `b`, `c`, `d` or `X`. A `-` character between two other characters forms a range that matches all characters from the first character to the second. For example, `[0-9]` matches any decimal digit. To include a literal `]` character, it must immediately follow the opening bracket `[`. To include a literal `-` character, it must be written first or last. Any character that does not have a defined special meaning inside a `[]` pair matches only itself.

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]'; -> 1
mysql> SELECT 'aXbc' REGEXP '^ [a-dXYZ]$'; -> 0
mysql> SELECT 'aXbc' REGEXP '^ [a-dXYZ]+$'; -> 1
mysql> SELECT 'aXbc' REGEXP '^ [^a-dXYZ]+$'; -> 0
mysql> SELECT 'gheis' REGEXP '^ [^a-dXYZ]+$'; -> 1
mysql> SELECT 'gheisa' REGEXP '^ [^a-dXYZ]+$'; -> 0
```

- `[.characters.]`

Within a bracket expression (written using `[` and `]`), matches the sequence of characters of that collating element. `characters` is either a single character or a character name like `newLine`. The following table lists the allowable character names.

The following table shows the allowable character names and the characters that they match. For characters given as numeric values, the values are represented in octal.

Name	Character	Name	Character
NUL	0	SOH	001
STX	002	ETX	003
EOT	004	ENQ	005
ACK	006	BEL	007
alert	007	BS	010
backspace	'\b'	HT	011
tab	'\t'	LF	012
newline	'\n'	VT	013
vertical-tab	'\v'	FF	014
form-feed	'\f'	CR	015
carriage-return	'\r'	SO	016
SI	017	DLE	020
DC1	021	DC2	022
DC3	023	DC4	024
NAK	025	SYN	026
ETB	027	CAN	030
EM	031	SUB	032
ESC	033	IS4	034
FS	034	IS3	035
GS	035	IS2	036
RS	036	IS1	037
US	037	space	' '
exclamation-mark	'!'	quotation-mark	'\"'
number-sign	'#'	dollar-sign	'\$'
percent-sign	'%'	ampersand	'&'
apostrophe	'\''	left-parenthesis	'('
right-parenthesis	')'	asterisk	'*'
plus-sign	'+'	comma	','
hyphen	'-'	hyphen-minus	'-'
period	'.'	full-stop	'.'
slash	'/'	solidus	'/'
zero	'0'	one	'1'
two	'2'	three	'3'
four	'4'	five	'5'
six	'6'	seven	'7'
eight	'8'	nine	'9'
colon	':'	semicolon	';'
less-than-sign	'<'	equals-sign	'='
greater-than-sign	'>'	question-mark	'?'
commercial-at	'@'	left-square-bracket	'['
backslash	'\"'	reverse-solidus	'\"'
right-square-bracket	']'	circumflex	'^'
circumflex-accent	'^'	underscore	'_'

low-line	'_'	grave-accent	'`'
left-brace	'{'	left-curly-bracket	'{'
vertical-line	' '	right-brace	'}'
right-curly-bracket	'}'	tilde	'~'
DEL	177		

```
mysql> SELECT '~' REGEXP '[[.~.]]';      -> 1
mysql> SELECT '~' REGEXP '[[.tilde.]]';  -> 1
```

- `[=character_class=]`

Within a bracket expression (written using `[` and `]`), `[=character_class=]` represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if `o` and `(+)` are the members of an equivalence class, then `[[=o=]]`, `[[=(+)=]]`, and `[o(+)]` are all synonymous. An equivalence class may not be used as an endpoint of a range.

- `[:character_class:]`

Within a bracket expression (written using `[` and `]`), `[:character_class:]` represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the [ctype\(3\)](#) manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

<code>alnum</code>	Alphanumeric characters
<code>alpha</code>	Alphabetic characters
<code>blank</code>	Whitespace characters
<code>cntrl</code>	Control characters
<code>digit</code>	Digit characters
<code>graph</code>	Graphic characters
<code>lower</code>	Lowercase alphabetic characters
<code>print</code>	Graphic or space characters
<code>punct</code>	Punctuation characters
<code>space</code>	Space, tab, newline, and carriage return
<code>upper</code>	Uppercase alphabetic characters
<code>xdigit</code>	Hexadecimal digit characters

```
mysql> SELECT 'justalnums' REGEXP '[[[:alnum:]]+];      -> 1
mysql> SELECT '!!!' REGEXP '[[[:alnum:]]+];            -> 0
```

- `[:<:]]`, `[:>:]]`

These markers stand for word boundaries. They match the beginning and end of words, respectively. A word is a sequence of word characters that is not preceded by or followed by word characters. A word character is an alphanumeric character in the `alnum` class or an underscore (`_`).

```
mysql> SELECT 'a word a' REGEXP '[:<:]word[:>:]]';    -> 1
mysql> SELECT 'a xword a' REGEXP '[:<:]word[:>:]]';    -> 0
```

To use a literal instance of a special character in a regular expression, precede it by two backslash (`\`) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string `1+2` that contains the special `+` character, only the last of the following regular expressions is the correct one:

```
mysql> SELECT '1+2' REGEXP '1+2';      -> 0
mysql> SELECT '1+2' REGEXP '1\+2';     -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';     -> 1
```

11.5. Numeric Functions

Name	Description
ABS()	Return the absolute value
ACOS()	Return the arc cosine
ASIN()	Return the arc sine
ATAN2(), ATAN()	Return the arc tangent of the two arguments
ATAN()	Return the arc tangent
/	Division operator
CEILING(), CEIL()	Return the smallest integer value not less than the argument
COS()	Return the cosine
COT()	Return the cotangent
CRC32() (v4.1.0)	Compute a cyclic redundancy check value
DEGREES()	Convert radians to degrees
DIV (v4.1.0)	Integer division
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
MOD()	Return the remainder
LN()	Return the natural logarithm of the argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
LOG()	Return the natural logarithm of the first argument
-	Minus operator
%	Modulo operator
PI()	Return the value of pi
+	Addition operator
POW(), POWER()	Return the argument raised to the specified power
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
*	Times operator
TRUNCATE()	Truncate to specified number of decimal places
-	Change the sign of the argument

11.5.1. Arithmetic Operators

Name	Description
/	Division operator
DIV (v4.1.0)	Integer division
-	Minus operator
%	Modulo operator
+	Addition operator

Name	Description
*	Times operator
-	Change the sign of the argument

The usual arithmetic operators are available. The precision of the result is determined according to the following rules:

- Note that in the case of `-`, `+`, and `*`, the result is calculated with `BIGINT` (64-bit) precision if both arguments are integers.
- If one of the arguments is an unsigned integer, and the other argument is also an integer, the result is an unsigned integer.
- If any of the operands of a `+`, `-`, `/`, `*`, `%` is a real or string value, then the precision of the result is the precision of the argument with the maximum precision.
- In multiplication and division, the precision of the result when using two exact values is the precision of the first argument + the value of the `div_precision_increment` global variable. For example, the expression `5.05 / 0.0014` would have a precision of six decimal places (`3607.142857`).

These rules are applied for each operation, such that nested calculations imply the precision of each component. Hence, `(14620 / 9432456) / (24250 / 9432456)`, would resolve first to `(0.0014) / (0.0026)`, with the final result having 8 decimal places (`0.57692308`).

Because of these rules and the method they are applied, care should be taken to ensure that components and sub-components of a calculation use the appropriate level of precision. See [Section 11.9, “Cast Functions and Operators”](#).

- `+`

Addition:

```
mysql> SELECT 3+5;
-> 8
```

- `-`

Subtraction:

```
mysql> SELECT 3-5;
-> -2
```

- `-`

Unary minus. This operator changes the sign of the argument.

```
mysql> SELECT - 2;
-> -2
```

Note

If this operator is used with a `BIGINT`, the return value is also a `BIGINT`. This means that you should avoid using `-` on integers that may have the value of -2^{63} .

- `*`

Multiplication:

```
mysql> SELECT 3*5;
-> 15
mysql> SELECT 18014398509481984*18014398509481984.0;
-> 324518553658426726783156020576256.0
mysql> SELECT 18014398509481984*18014398509481984;
-> 0
```


The result of the last expression is incorrect because the result of the integer multiplication exceeds the 64-bit range of **BIGINT** calculations. (See [Section 10.2, “Numeric Types”](#).)

- /

Division:

```
mysql> SELECT 3/5;
-> 0.60
```

Division by zero produces a **NULL** result:

```
mysql> SELECT 102/(1-1);
-> NULL
```

A division is calculated with **BIGINT** arithmetic only if performed in a context where its result is converted to an integer.

- DIV

Integer division. Similar to **FLOOR()**, but is safe with **BIGINT** values.

```
mysql> SELECT 5 DIV 2;
-> 2
```

- N % M

Modulo operation. Returns the remainder of *N* divided by *M*. For more information, see the description for the **MOD()** function in [Section 11.5.2, “Mathematical Functions”](#).

11.5.2. Mathematical Functions

Name	Description
ABS()	Return the absolute value
ACOS()	Return the arc cosine
ASIN()	Return the arc sine
ATAN2(), ATAN()	Return the arc tangent of the two arguments
ATAN()	Return the arc tangent
CEILING(), CEIL()	Return the smallest integer value not less than the argument
COS()	Return the cosine
COT()	Return the cotangent
CRC32() (v4.1.0)	Compute a cyclic redundancy check value
DEGREES()	Convert radians to degrees
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
MOD()	Return the remainder
LN()	Return the natural logarithm of the argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
LOG()	Return the natural logarithm of the first argument
PI()	Return the value of pi
POW(), POWER()	Return the argument raised to the specified power
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value

Name	Description
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
TRUNCATE()	Truncate to specified number of decimal places

All mathematical functions return **NULL** in the event of an error.

- [ABS\(X\)](#)

Returns the absolute value of *X*.

```
mysql> SELECT ABS(2);
-> 2
mysql> SELECT ABS(-32);
-> 32
```

This function is safe to use with **BIGINT** values.

- [ACOS\(X\)](#)

Returns the arc cosine of *X*, that is, the value whose cosine is *X*. Returns **NULL** if *X* is not in the range **-1** to **1**.

```
mysql> SELECT ACOS(1);
-> 0
mysql> SELECT ACOS(1.0001);
-> NULL
mysql> SELECT ACOS(0);
-> 1.5707963267949
```

- [ASIN\(X\)](#)

Returns the arc sine of *X*, that is, the value whose sine is *X*. Returns **NULL** if *X* is not in the range **-1** to **1**.

```
mysql> SELECT ASIN(0.2);
-> 0.20135792079033
mysql> SELECT ASIN('foo');
```

```
+-----+
| ASIN('foo') |
+-----+
| 0 |
+-----+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> SHOW WARNINGS;
```

```
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
+-----+-----+-----+
```

- [ATAN\(X\)](#)

Returns the arc tangent of *X*, that is, the value whose tangent is *X*.

```
mysql> SELECT ATAN(2);
-> 1.1071487177941
mysql> SELECT ATAN(-2);
-> -1.1071487177941
```

- [ATAN\(Y,X\)](#), [ATAN2\(Y,X\)](#)

Returns the arc tangent of the two variables *X* and *Y*. It is similar to calculating the arc tangent of *Y* / *X*, except that the signs of

both arguments are used to determine the quadrant of the result.

```
mysql> SELECT ATAN(-2,2);
-> -0.78539816339745
mysql> SELECT ATAN2(PI(),0);
-> 1.5707963267949
```

- **CEILING(*X*)**, **CEIL(*X*)**

Returns the smallest integer value not less than *X*. **CEILING()** and **CEIL()** are synonymous.

```
mysql> SELECT CEILING(1.23);
-> 2
mysql> SELECT CEIL(-1.23);
-> -1
```

For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- **COS(*X*)**

Returns the cosine of *X*, where *X* is given in radians.

```
mysql> SELECT COS(PI());
-> -1
```

- **COT(*X*)**

Returns the cotangent of *X*.

```
mysql> SELECT COT(12);
-> -1.5726734063977
mysql> SELECT COT(0);
-> NULL
```

- **CRC32(*expr*)**

Computes a cyclic redundancy check value and returns a 32-bit unsigned value. The result is **NULL** if the argument is **NULL**. The argument is expected to be a string and (if possible) is treated as one if it is not.

```
mysql> SELECT CRC32('MySQL');
-> 3259397556
mysql> SELECT CRC32('mysql');
-> 2501908538
```

- **DEGREES(*X*)**

Returns the argument *X*, converted from radians to degrees.

```
mysql> SELECT DEGREES(PI());
-> 180
mysql> SELECT DEGREES(PI() / 2);
-> 90
```

- **EXP(*X*)**

Returns the value of *e* (the base of natural logarithms) raised to the power of *X*.

```
mysql> SELECT EXP(2);
-> 7.3890560989307
mysql> SELECT EXP(-2);
-> 0.13533528323661
mysql> SELECT EXP(0);
-> 1
```

- **FLOOR(*X*)**

Returns the largest integer value not greater than *X*.

```
mysql> SELECT FLOOR(1.23);
-> 1
mysql> SELECT FLOOR(-1.23);
-> -2
```

For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- **FORMAT(*X*, *D*)**

Formats the number *X* to a format like '*#, ###, ###.##*', rounded to *D* decimal places, and returns the result as a string. For details, see [Section 11.4, “String Functions”](#).

- **LN(*X*)**

Returns the natural logarithm of *X*; that is, the base-*e* logarithm of *X*.

```
mysql> SELECT LN(2);
-> 0.69314718055995
mysql> SELECT LN(-2);
-> NULL
```

This function is synonymous with **LOG(*X*)**.

- **LOG(*X*), LOG(*B*, *X*)**

If called with one parameter, this function returns the natural logarithm of *X*.

```
mysql> SELECT LOG(2);
-> 0.69314718055995
mysql> SELECT LOG(-2);
-> NULL
```

If called with two parameters, this function returns the logarithm of *X* for an arbitrary base *B*.

```
mysql> SELECT LOG(2, 65536);
-> 16
mysql> SELECT LOG(10, 100);
-> 2
```

LOG(*B*, *X*) is equivalent to **LOG(*X*) / LOG(*B*)**.

- **LOG2(*X*)**

Returns the base-2 logarithm of *X*.

```
mysql> SELECT LOG2(65536);
-> 16
mysql> SELECT LOG2(-100);
-> NULL
```

LOG2() is useful for finding out how many bits a number requires for storage. This function is equivalent to the expression **LOG(*X*) / LOG(2)**.

- **LOG10(*X*)**

Returns the base-10 logarithm of *X*.

```
mysql> SELECT LOG10(2);
-> 0.30102999566398
mysql> SELECT LOG10(100);
-> 2
mysql> SELECT LOG10(-100);
-> NULL
```

LOG10(*X*) is equivalent to **LOG(10, *X*)**.

- **MOD(*N*, *M*), *N* % *M*, *N* MOD *M***

Modulo operation. Returns the remainder of N divided by M .

```
mysql> SELECT MOD(234, 10);
-> 4
mysql> SELECT 253 % 7;
-> 1
mysql> SELECT MOD(29,9);
-> 2
mysql> SELECT 29 MOD 9;
-> 2
```

This function is safe to use with **BIGINT** values.

MOD() also works on values that have a fractional part and returns the exact remainder after division:

```
mysql> SELECT MOD(34.5,3);
-> 1.5
```

MOD(N , 0) returns **NULL**.

- **PI()**

Returns the value of π (pi). The default number of decimal places displayed is seven, but MySQL uses the full double-precision value internally.

```
mysql> SELECT PI();
-> 3.141593
mysql> SELECT PI()+0.00000000000000000000;
-> 3.141592653589793116
```

- **POW(X , Y), POWER(X , Y)**

Returns the value of X raised to the power of Y .

```
mysql> SELECT POW(2,2);
-> 4
mysql> SELECT POW(2, -2);
-> 0.25
```

- **RADIANS(X)**

Returns the argument X , converted from degrees to radians. (Note that π radians equals 180 degrees.)

```
mysql> SELECT RADIANS(90);
-> 1.5707963267949
```

- **RAND(), RAND(N)**

Returns a random floating-point value v in the range $0 \leq v < 1.0$. If a constant integer argument N is specified, it is used as the seed value, which produces a repeatable sequence of column values.

```
mysql> SELECT RAND();
-> 0.9233482386203
mysql> SELECT RAND(20);
-> 0.15888261251047
mysql> SELECT RAND(20);
-> 0.15888261251047
mysql> SELECT RAND();
-> 0.63553050033332
mysql> SELECT RAND();
-> 0.70100469486881
mysql> SELECT RAND(20);
-> 0.15888261251047
```

With a constant initializer, the seed is initialized once when the statement is compiled, prior to execution. As of MySQL 5.1.16, if a non-constant initializer (such as a column name) is used as the argument, the seed is initialized with the value for each invocation of **RAND()**. (One implication of this is that for equal argument values, **RAND()** will return the same value each time.) From MySQL 5.1.3 to 5.1.15, non-constant arguments are disallowed. Before that, the effect of using a non-constant argument is undefined.

To obtain a random integer R in the range $i \leq R < j$, use the expression `FLOOR( $i$  + RAND() * ( $j$  -  $i$ ))`. For example, to obtain a random integer in the range the range $7 \leq R < 12$, you could use the following statement:

```
SELECT FLOOR(7 + (RAND() * 5));
```

You cannot use a column with `RAND()` values in an `ORDER BY` clause, because `ORDER BY` would evaluate the column multiple times. However, you can retrieve rows in random order like this:

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```

`ORDER BY RAND()` combined with `LIMIT` is useful for selecting a random sample from a set of rows:

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d -> ORDER BY RAND() LIMIT 1000;
```

Note that `RAND()` in a `WHERE` clause is re-evaluated every time the `WHERE` is executed.

`RAND()` is not meant to be a perfect random generator, but instead is a fast way to generate *ad hoc* random numbers which is portable between platforms for the same MySQL version.

- **ROUND(X), ROUND(X , D)**

Rounds the argument X to D decimal places. The rounding algorithm depends on the data type of X . D defaults to 0 if not specified. D can be negative to cause D digits left of the decimal point of the value X to become zero.

```
mysql> SELECT ROUND(-1.23);
-> -1
mysql> SELECT ROUND(-1.58);
-> -2
mysql> SELECT ROUND(1.58);
-> 2
mysql> SELECT ROUND(1.298, 1);
-> 1.3
mysql> SELECT ROUND(1.298, 0);
-> 1
mysql> SELECT ROUND(23.298, -1);
-> 20
```

The return type is the same type as that of the first argument (assuming that it is integer, double, or decimal). This means that for an integer argument, the result is an integer (no decimal places):

```
mysql> SELECT ROUND(150.000,2), ROUND(150,2);
+-----+-----+
| ROUND(150.000,2) | ROUND(150,2) |
+-----+-----+
| 150.00 | 150 |
+-----+-----+
```

`ROUND()` uses the precision math library for exact-value arguments when the first argument is a decimal value:

- For exact-value numbers, `ROUND()` uses the “round half up” or “round toward nearest” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` uses the “round to nearest even” rule: A value with any fractional part is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+-----+-----+
| ROUND(2.5) | ROUND(25E-1) |
+-----+-----+
| 3 | 2 |
+-----+-----+
```

For more information, see [Chapter 24, Precision Math](#).

- **SIGN(*X*)**

Returns the sign of the argument as **-1**, **0**, or **1**, depending on whether *X* is negative, zero, or positive.

```
mysql> SELECT SIGN(-32);
-> -1
mysql> SELECT SIGN(0);
-> 0
mysql> SELECT SIGN(234);
-> 1
```

- **SIN(*X*)**

Returns the sine of *X*, where *X* is given in radians.

```
mysql> SELECT SIN(PI());
-> 1.2246063538224e-16
mysql> SELECT ROUND(SIN(PI()));
-> 0
```

- **SQRT(*X*)**

Returns the square root of a non-negative number *X*.

```
mysql> SELECT SQRT(4);
-> 2
mysql> SELECT SQRT(20);
-> 4.4721359549996
mysql> SELECT SQRT(-16);
-> NULL
```

- **TAN(*X*)**

Returns the tangent of *X*, where *X* is given in radians.

```
mysql> SELECT TAN(PI());
-> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
-> 1.5574077246549
```

- **TRUNCATE(*X*, *D*)**

Returns the number *X*, truncated to *D* decimal places. If *D* is **0**, the result has no decimal point or fractional part. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

```
mysql> SELECT TRUNCATE(1.223,1);
-> 1.2
mysql> SELECT TRUNCATE(1.999,1);
-> 1.9
mysql> SELECT TRUNCATE(1.999,0);
-> 1
mysql> SELECT TRUNCATE(-1.999,1);
-> -1.9
mysql> SELECT TRUNCATE(122,-2);
-> 100
mysql> SELECT TRUNCATE(10.28*100,0);
-> 1028
```

All numbers are rounded toward zero.

11.6. Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See [Section 10.3, “Date and Time Types”](#), for a description of the range of values each date and time type has and the valid formats in which values may be specified.

Name	Description
ADDDATE() (v4.1.1)	Add dates
ADDTIME() (v4.1.1)	Add time

Name	Description
CONVERT_TZ() (v4.1.3)	Convert from one timezone to another
CURDATE()	Return the current date
CURRENT_DATE() , CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME() , CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP() , CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time
DATE_ADD()	Add two dates
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract two dates
DATE() (v4.1.1)	Extract the date part of a date or datetime expression
DATEDIFF() (v4.1.1)	Subtract two dates
DAY() (v4.1.1)	Synonym for DAYOFMONTH()
DAYNAME() (v4.1.21)	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (1-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format date as a UNIX timestamp
GET_FORMAT() (v4.1.1)	Return a date format string
HOUR()	Extract the hour
LAST_DAY (v4.1.1)	Return the last day of the month for the argument
LOCALTIME() , LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP , LOCALTIMESTAMP() (v4.0.6)	Synonym for NOW()
MAKEDATE() (v4.1.1)	Create a date from the year and day of year
MAKETIME (v4.1.1)	MAKETIME()
MICROSECOND() (v4.1.1)	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME() (v4.1.21)	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
STR_TO_DATE() (v4.1.1)	Convert a string to a date
SUBDATE()	When invoked with three arguments a synonym for DATE_SUB()
SUBTIME() (v4.1.1)	Subtract times
SYSDATE()	Return the time at which the function executes
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIME() (v4.1.1)	Extract the time portion of the expression passed

Name	Description
TIMEDIFF() (v4.1.1)	Subtract time
TIMESTAMP() (v4.1.1)	With a single argument, this function returns the date or datetime expression. With two arguments, the sum of the arguments
TIMESTAMPADD() (v5.0.0)	Add an interval to a datetime expression
TIMESTAMPDIFF() (v5.0.0)	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
UNIX_TIMESTAMP()	Return a UNIX timestamp
UTC_DATE() (v4.1.1)	Return the current UTC date
UTC_TIME() (v4.1.1)	Return the current UTC time
UTC_TIMESTAMP() (v4.1.1)	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR() (v4.1.1)	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

Here is an example that uses date functions. The following query selects all rows with a [date_col](#) value from within the last 30 days:

```
mysql> SELECT something FROM tbl_name
-> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

Note that the query also selects rows with dates that lie in the future.

Functions that expect date values usually accept datetime values and ignore the time part. Functions that expect time values usually accept datetime values and ignore the date part.

Functions that return the current date or time each are evaluated only once per query at the start of query execution. This means that multiple references to a function such as [NOW\(\)](#) within a single query always produce the same result (for our purposes a single query also includes a call to a stored routine or trigger and all sub-routines called by that routine/trigger). This principle also applies to [CURDATE\(\)](#), [CURTIME\(\)](#), [UTC_DATE\(\)](#), [UTC_TIME\(\)](#), [UTC_TIMESTAMP\(\)](#), and to any of their synonyms.

The [CURRENT_TIMESTAMP\(\)](#), [CURRENT_TIME\(\)](#), [CURRENT_DATE\(\)](#), and [FROM_UNIXTIME\(\)](#) functions return values in the connection's current time zone, which is available as the value of the [time_zone](#) system variable. In addition, [UNIX_TIMESTAMP\(\)](#) assumes that its argument is a datetime value in the current time zone. See [Section 5.10.8, “MySQL Server Time Zone Support”](#).

Some date functions can be used with “zero” dates or incomplete dates such as '2001-11-00', whereas others cannot. Functions that extract parts of dates typically work with incomplete dates. For example:

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
-> 0, 0
```

Other functions expect complete dates and return [NULL](#) for incomplete dates. These include functions that perform date arithmetic or that map parts of dates to names. For example:

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
-> NULL
mysql> SELECT DAYNAME('2006-05-00');
-> NULL
```

- [ADDDATE\(date,INTERVAL expr unit\)](#), [ADDDATE\(expr,days\)](#)

When invoked with the [INTERVAL](#) form of the second argument, [ADDDATE\(\)](#) is a synonym for [DATE_ADD\(\)](#). The related function [SUBDATE\(\)](#) is a synonym for [DATE_SUB\(\)](#). For information on the [INTERVAL unit](#) argument, see the discussion for [DATE_ADD\(\)](#).

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
-> '1998-02-02'
```

```
mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
-> '1998-02-02'
```

When invoked with the *days* form of the second argument, MySQL treats it as an integer number of days to be added to *expr*.

```
mysql> SELECT ADDDATE('1998-01-02', 31);
-> '1998-02-02'
```

- **ADDTIME(*expr1*,*expr2*)**

ADDTIME() adds *expr2* to *expr1* and returns the result. *expr1* is a time or datetime expression, and *expr2* is a time expression.

```
mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999',
-> '1 1:1:1.000002');
-> '1998-01-02 01:01:01.000001'
mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
-> '03:00:01.999997'
```

- **CONVERT_TZ(*dt*,*from_tz*,*to_tz*)**

CONVERT_TZ() converts a datetime value *dt* from the time zone given by *from_tz* to the time zone given by *to_tz* and returns the resulting value. Time zones are specified as described in [Section 5.10.8, “MySQL Server Time Zone Support”](#). This function returns **NULL** if the arguments are invalid.

If the value falls out of the supported range of the **TIMESTAMP** type when converted from *from_tz* to UTC, no conversion occurs. The **TIMESTAMP** range is described in [Section 10.1.2, “Overview of Date and Time Types”](#).

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00', 'GMT', 'MET');
-> '2004-01-01 13:00:00'
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00', '+00:00', '+10:00');
-> '2004-01-01 22:00:00'
```

Note

To use named time zones such as **'MET'** or **'Europe/Moscow'**, the time zone tables must be properly set up. See [Section 5.10.8, “MySQL Server Time Zone Support”](#), for instructions.

Before MySQL 5.1.17, if you intend to use **CONVERT_TZ()** while other tables are locked with **LOCK TABLES**, you must also lock the **mysql.time_zone_name** table. See [Section 12.4.5, “LOCK TABLES and UNLOCK TABLES Syntax”](#).

- **CURDATE()**

Returns the current date as a value in **'YYYY-MM-DD'** or **YYYYMMDD** format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURDATE();
-> '1997-12-15'
mysql> SELECT CURDATE() + 0;
-> 19971215
```

- **CURRENT_DATE, CURRENT_DATE()**

CURRENT_DATE and **CURRENT_DATE()** are synonyms for **CURDATE()**.

- **CURTIME()**

Returns the current time as a value in **'HH:MM:SS'** or **HHMMSS** format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT CURTIME();
-> '23:50:26'
mysql> SELECT CURTIME() + 0;
-> 235026
```

- **CURRENT_TIME, CURRENT_TIME()**

`CURRENT_TIME` and `CURRENT_TIME()` are synonyms for `CURTIME()`.

- `CURRENT_TIMESTAMP`, `CURRENT_TIMESTAMP()`

`CURRENT_TIMESTAMP` and `CURRENT_TIMESTAMP()` are synonyms for `NOW()`.

- `DATE(expr)`

Extracts the date part of the date or datetime expression *expr*.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
-> '2003-12-31'
```

- `DATEDIFF(expr1,expr2)`

`DATEDIFF()` returns *expr1* – *expr2* expressed as a value in days from one date to the other. *expr1* and *expr2* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
-> 1
mysql> SELECT DATEDIFF('1997-11-30 23:59:59','1997-12-31');
-> -31
```

- `DATE_ADD(date,INTERVAL expr unit)`, `DATE_SUB(date,INTERVAL expr unit)`

These functions perform date arithmetic. *date* is a `DATETIME` or `DATE` value specifying the starting date. *expr* is an expression specifying the interval value to be added or subtracted from the starting date. *expr* is a string; it may start with a '-' for negative intervals. *unit* is a keyword indicating the units in which the expression should be interpreted.

The `INTERVAL` keyword and the *unit* specifier are not case sensitive.

The following table shows the expected form of the *expr* argument for each *unit* value.

<i>unit</i> Value	Expected <i>expr</i> Format
<code>MICROSECOND</code>	<code>MICROSECONDS</code>
<code>SECOND</code>	<code>SECONDS</code>
<code>MINUTE</code>	<code>MINUTES</code>
<code>HOURL</code>	<code>HOURS</code>
<code>DAY</code>	<code>DAYS</code>
<code>WEEK</code>	<code>WEEKS</code>
<code>MONTH</code>	<code>MONTHS</code>
<code>QUARTER</code>	<code>QUARTERS</code>
<code>YEAR</code>	<code>YEARS</code>
<code>SECOND_MICROSECOND</code>	<code>'SECONDS.MICROSECONDS'</code>
<code>MINUTE_MICROSECOND</code>	<code>'MINUTES.MICROSECONDS'</code>
<code>MINUTE_SECOND</code>	<code>'MINUTES:SECONDS'</code>
<code>HOURL_MICROSECOND</code>	<code>'HOURS.MICROSECONDS'</code>
<code>HOURL_SECOND</code>	<code>'HOURS:MINUTES:SECONDS'</code>
<code>HOURL_MINUTE</code>	<code>'HOURS:MINUTES'</code>
<code>DAY_MICROSECOND</code>	<code>'DAYS.MICROSECONDS'</code>
<code>DAY_SECOND</code>	<code>'DAYS HOURS:MINUTES:SECONDS'</code>
<code>DAY_MINUTE</code>	<code>'DAYS HOURS:MINUTES'</code>
<code>DAY_HOUR</code>	<code>'DAYS HOURS'</code>
<code>YEAR_MONTH</code>	<code>'YEARS-MONTHS'</code>

MySQL allows any punctuation delimiter in the *expr* format. Those shown in the table are the suggested delimiters. If the *date* argument is a *DATE* value and your calculations involve only *YEAR*, *MONTH*, and *DAY* parts (that is, no time parts), the result is a *DATE* value. Otherwise, the result is a *DATETIME* value.

Date arithmetic also can be performed using *INTERVAL* together with the *+* or *-* operator:

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

INTERVAL expr unit is allowed on either side of the *+* operator if the expression on the other side is a date or datetime value. For the *-* operator, *INTERVAL expr unit* is allowed only on the right side, because it makes no sense to subtract a date or datetime value from an interval.

```
mysql> SELECT '1997-12-31 23:59:59' + INTERVAL 1 SECOND;
-> '1998-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '1997-12-31';
-> '1998-01-01'
mysql> SELECT '1998-01-01' - INTERVAL 1 SECOND;
-> '1997-12-31 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL 1 SECOND);
-> '1998-01-01 00:00:00'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL 1 DAY);
-> '1998-01-01 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL '1:1' MINUTE_SECOND);
-> '1998-01-01 00:01:00'
mysql> SELECT DATE_SUB('1998-01-01 00:00:00',
-> INTERVAL '1 1:1:1' DAY_SECOND);
-> '1997-12-30 22:58:59'
mysql> SELECT DATE_ADD('1998-01-01 00:00:00',
-> INTERVAL '-1 10' DAY_HOUR);
-> '1997-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
-> INTERVAL '1.999999' SECOND_MICROSECOND);
-> '1993-01-01 00:00:01.000001'
```

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the *unit* keyword), MySQL assumes that you have left out the leftmost parts of the interval value. For example, if you specify a *unit* of *DAY_SECOND*, the value of *expr* is expected to have days, hours, minutes, and seconds parts. If you specify a value like *'1:10'*, MySQL assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, *'1:10' DAY_SECOND* is interpreted in such a way that it is equivalent to *'1:10' MINUTE_SECOND*. This is analogous to the way that MySQL interprets *TIME* values as representing elapsed time rather than as a time of day.

If you add to or subtract from a date value something that contains a time part, the result is automatically converted to a datetime value:

```
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 DAY);
-> '1999-01-02'
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
-> '1999-01-01 01:00:00'
```

If you add *MONTH*, *YEAR_MONTH*, or *YEAR* and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month:

```
mysql> SELECT DATE_ADD('1998-01-30', INTERVAL 1 MONTH);
-> '1998-02-28'
```

Date arithmetic operations require complete dates and do not work with incomplete dates such as *'2006-07-00'* or badly malformed dates:

```
mysql> SELECT DATE_ADD('2006-07-00', INTERVAL 1 DAY);
-> NULL
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;
-> NULL
```

- *DATE_FORMAT(date, format)*

Formats the *date* value according to the *format* string.

The following specifiers may be used in the *format* string. The ‘%’ character is required before format specifier characters.

Specifier	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, ...)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week
%u	Week (00..53), where Monday is the first day of the week
%V	Week (01..53), where Sunday is the first day of the week; used with %X
%v	Week (01..53), where Monday is the first day of the week; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal ‘%’ character
%x	x, for any ‘x’ not listed above

Ranges for the month and day specifiers begin with zero due to the fact that MySQL allows the storing of incomplete dates such as '2004-00-00'.

As of MySQL 5.1.12, the language used for day and month names and abbreviations is controlled by the value of the `lc_time_names` system variable (Section 5.10.9, “MySQL Server Locale Support”).

As of MySQL 5.1.15, `DATE_FORMAT()` returns a string with a character set and collation given by `character_set_connection` and `collation_connection` so that it can return month and weekday names containing non-ASCII characters. Before 5.1.15, the return value is a binary string.

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
-> 'Saturday October 1997'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
-> '22:23:00'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
                          '%D %y %a %d %m %b %j');
-> '4th 97 Sat 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
                          '%H %k %I %r %T %S %w');
-> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
-> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
-> '00'
```

- `DATE_SUB(date, INTERVAL expr unit)`

See `DATE_ADD()`.

- `DAY(date)`

`DAY()` is a synonym for `DAYOFMONTH()`.

- `DAYNAME(date)`

Returns the name of the weekday for *date*. As of MySQL 5.1.12, the language used for the name is controlled by the value of the `lc_time_names` system variable (Section 5.10.9, “MySQL Server Locale Support”).

```
mysql> SELECT DAYNAME('1998-02-05');
-> 'Thursday'
```

- `DAYOFMONTH(date)`

Returns the day of the month for *date*, in the range 0 to 31.

```
mysql> SELECT DAYOFMONTH('1998-02-03');
-> 3
```

- `DAYOFWEEK(date)`

Returns the weekday index for *date* (1 = Sunday, 2 = Monday, ..., 7 = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('1998-02-03');
-> 3
```

- `DAYOFYEAR(date)`

Returns the day of the year for *date*, in the range 1 to 366.

```
mysql> SELECT DAYOFYEAR('1998-02-03');
-> 34
```

- `EXTRACT(unit FROM date)`

The `EXTRACT()` function uses the same kinds of unit specifiers as `DATE_ADD()` or `DATE_SUB()`, but extracts parts from the date rather than performing date arithmetic.

```
mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');
-> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');
-> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '1999-07-02 01:02:03');
-> 20102
mysql> SELECT EXTRACT(MICROSECOND
                      FROM '2003-01-02 10:30:00.000123');
-> 123
```

- `FROM_DAYS(N)`

Given a day number *N*, returns a **DATE** value.

```
mysql> SELECT FROM_DAYS(729669);
-> '1997-10-07'
```

Use **FROM_DAYS()** with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582). See [Section 11.7, “What Calendar Is Used By MySQL?”](#).

- **FROM_UNIXTIME(*unix_timestamp*), FROM_UNIXTIME(*unix_timestamp*, *format*)**

Returns a representation of the *unix_timestamp* argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone. *unix_timestamp* is an internal timestamp value such as is produced by the **UNIX_TIMESTAMP()** function.

If *format* is given, the result is formatted according to the *format* string, which is used the same way as listed in the entry for the **DATE_FORMAT()** function.

```
mysql> SELECT FROM_UNIXTIME(875996580);
-> '1997-10-04 22:23:00'
mysql> SELECT FROM_UNIXTIME(875996580) + 0;
-> 19971004222300
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
-> '%Y %D %M %h:%i:%s %x');
-> '2003 6th August 06:22:58 2003'
```

Note: If you use **UNIX_TIMESTAMP()** and **FROM_UNIXTIME()** to convert between **TIMESTAMP** values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For details, see the description of the **UNIX_TIMESTAMP()** function.

- **GET_FORMAT(DATE|TIME|DATETIME, 'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')**

Returns a format string. This function is useful in combination with the **DATE_FORMAT()** and the **STR_TO_DATE()** functions.

The possible values for the first and second arguments result in several possible format strings (for the specifiers used, see the table in the **DATE_FORMAT()** function description). ISO format refers to ISO 9075, not ISO 8601.

Function Call	Result
GET_FORMAT(DATE, 'USA')	'%m.%d.%Y'
GET_FORMAT(DATE, 'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE, 'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE, 'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE, 'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME, 'USA')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME, 'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME, 'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME, 'EUR')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME, 'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME, 'USA')	'%h:%i:%s %p'
GET_FORMAT(TIME, 'JIS')	'%H:%i:%s'
GET_FORMAT(TIME, 'ISO')	'%H:%i:%s'
GET_FORMAT(TIME, 'EUR')	'%H.%i.%s'
GET_FORMAT(TIME, 'INTERNAL')	'%H%i%s'

TIMESTAMP can also be used as the first argument to **GET_FORMAT()**, in which case the function returns the same values as for **DATETIME**.

```
mysql> SELECT DATE_FORMAT('2003-10-03', GET_FORMAT(DATE, 'EUR'));
-> '03.10.2003'
```

```
mysql> SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
-> '2003-10-31'
```

- **HOUR(*time*)**

Returns the hour for *time*. The range of the return value is 0 to 23 for time-of-day values. However, the range of **TIME** values actually is much larger, so **HOUR** can return values greater than 23.

```
mysql> SELECT HOUR('10:05:03');
-> 10
mysql> SELECT HOUR('272:59:59');
-> 272
```

- **LAST_DAY(*date*)**

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns **NULL** if the argument is invalid.

```
mysql> SELECT LAST_DAY('2003-02-05');
-> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
-> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
-> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
-> NULL
```

- **LOCALTIME, LOCALTIME()**

LOCALTIME and **LOCALTIME()** are synonyms for **NOW()**.

- **LOCALTIMESTAMP, LOCALTIMESTAMP()**

LOCALTIMESTAMP and **LOCALTIMESTAMP()** are synonyms for **NOW()**.

- **MAKEDATE(*year*,*dayofyear*)**

Returns a date, given year and day-of-year values. *dayofyear* must be greater than 0 or the result is **NULL**.

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);
-> '2001-01-31', '2001-02-01'
mysql> SELECT MAKEDATE(2001,365), MAKEDATE(2004,365);
-> '2001-12-31', '2004-12-30'
mysql> SELECT MAKEDATE(2001,0);
-> NULL
```

- **MAKETIME(*hour*,*minute*,*second*)**

Returns a time value calculated from the *hour*, *minute*, and *second* arguments.

```
mysql> SELECT MAKETIME(12,15,30);
-> '12:15:30'
```

- **MICROSECOND(*expr*)**

Returns the microseconds from the time or datetime expression *expr* as a number in the range from 0 to 999999.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
-> 123456
mysql> SELECT MICROSECOND('1997-12-31 23:59:59.000010');
-> 10
```

- **MINUTE(*time*)**

Returns the minute for *time*, in the range 0 to 59.

```
mysql> SELECT MINUTE('98-02-03 10:05:03');
-> 5
```


- **MONTH(*date*)**

Returns the month for *date*, in the range 0 to 12.

```
mysql> SELECT MONTH('1998-02-03');
-> 2
```

- **MONTHNAME(*date*)**

Returns the full name of the month for *date*. As of MySQL 5.1.12, the language used for the name is controlled by the value of the `lc_time_names` system variable (Section 5.10.9, “MySQL Server Locale Support”).

```
mysql> SELECT MONTHNAME('1998-02-05');
-> 'February'
```

- **NOW()**

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT NOW();
-> '1997-12-15 23:50:26'
mysql> SELECT NOW() + 0;
-> 19971215235026
```

NOW() returns a constant time that indicates the time at which the statement began to execute. (Within a stored routine or trigger, **NOW()** returns the time at which the routine or triggering statement began to execute.) This differs from the behavior for **SYS-DATE()**, which returns the exact time at which it executes.

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW()          | SLEEP(2) | NOW()          |
+-----+-----+-----+
| 2006-04-12 13:47:36 |          0 | 2006-04-12 13:47:36 |
+-----+-----+-----+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE()      | SLEEP(2) | SYSDATE()      |
+-----+-----+-----+
| 2006-04-12 13:47:44 |          0 | 2006-04-12 13:47:46 |
+-----+-----+-----+
```

See the description for **SYS-DATE()** for additional information about the differences between the two functions.

- **PERIOD_ADD(*P*, *N*)**

Adds *N* months to period *P* (in the format **YYMM** or **YYYYMM**). Returns a value in the format **YYYYMM**. Note that the period argument *P* is *not* a date value.

```
mysql> SELECT PERIOD_ADD(9801,2);
-> 199803
```

- **PERIOD_DIFF(*P1*, *P2*)**

Returns the number of months between periods *P1* and *P2*. *P1* and *P2* should be in the format **YYMM** or **YYYYMM**. Note that the period arguments *P1* and *P2* are *not* date values.

```
mysql> SELECT PERIOD_DIFF(9802,199703);
-> 11
```

- **QUARTER(*date*)**

Returns the quarter of the year for *date*, in the range 1 to 4.

```
mysql> SELECT QUARTER('98-04-01');
-> 2
```

- `SECOND(time)`

Returns the second for *time*, in the range 0 to 59.

```
mysql> SELECT SECOND('10:05:03');
-> 3
```

- `SEC_TO_TIME(seconds)`

Returns the *seconds* argument, converted to hours, minutes, and seconds, as a `TIME` value. The range of the result is constrained to that of the `TIME` data type. A warning occurs if the argument corresponds to a value outside that range.

```
mysql> SELECT SEC_TO_TIME(2378);
-> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
-> 3938
```

- `STR_TO_DATE(str, format)`

This is the inverse of the `DATE_FORMAT()` function. It takes a string *str* and a format string *format*. `STR_TO_DATE()` returns a `DATETIME` value if the format string contains both date and time parts, or a `DATE` or `TIME` value if the string contains only date or time parts.

The date, time, or datetime values contained in *str* should be given in the format indicated by *format*. For the specifiers that can be used in *format*, see the `DATE_FORMAT()` function description. If *str* contains an illegal date, time, or datetime value, `STR_TO_DATE()` returns `NULL`. An illegal value also produces a warning.

Range checking on the parts of date values is as described in [Section 10.3.1, “The DATETIME, DATE, and TIMESTAMP Types”](#). This means, for example, that “zero” dates or dates with part values of 0 are allowed unless the SQL mode is set to disallow such values.

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
-> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
-> '2004-04-31'
```

Note

You cannot use format `"%X%V"` to convert a year-week string to a date because the combination of a year and week does not uniquely identify a year and month if the week crosses a month boundary. To convert a year-week to a date, then you should also specify the weekday:

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
-> '2004-10-18'
```

- `SUBDATE(date, INTERVAL expr unit), SUBDATE(expr, days)`

When invoked with the `INTERVAL` form of the second argument, `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL unit` argument, see the discussion for `DATE_ADD()`.

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
```

The second form allows the use of an integer value for *days*. In such cases, it is interpreted as the number of days to be subtracted from the date or datetime expression *expr*.

```
mysql> SELECT SUBDATE('1998-01-02 12:00:00', 31);
-> '1997-12-02 12:00:00'
```

- `SUBTIME(expr1, expr2)`

`SUBTIME()` returns *expr1* – *expr2* expressed as a value in the same format as *expr1*. *expr1* is a time or datetime expression, and *expr2* is a time expression.

```
mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999', '1 1:1:1.000002');
```

```
mysql> -> '1997-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
mysql> -> '-00:59:59.999999'
```

- **`SYSDATE()`**

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

`SYSDATE()` returns the time at which it executes. This differs from the behavior for **`NOW()`**, which returns a constant time that indicates the time at which the statement began to execute. (Within a stored routine or trigger, **`NOW()`** returns the time at which the routine or triggering statement began to execute.)

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW() | SLEEP(2) | NOW() |
+-----+-----+-----+
| 2006-04-12 13:47:36 | 0 | 2006-04-12 13:47:36 |
+-----+-----+-----+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE() | SLEEP(2) | SYSDATE() |
+-----+-----+-----+
| 2006-04-12 13:47:44 | 0 | 2006-04-12 13:47:46 |
+-----+-----+-----+
```

In addition, the **`SET TIMESTAMP`** statement affects the value returned by **`NOW()`** but not by **`SYSDATE()`**. This means that timestamp settings in the binary log have no effect on invocations of **`SYSDATE()`**.

Because **`SYSDATE()`** can return different values even within the same statement, and is not affected by **`SET TIMESTAMP`**, it is non-deterministic and therefore unsafe for replication if statement-based binary logging is used. If that is a problem, you can use row-based logging, or start the server with the **`--sysdate-is-now`** option to cause **`SYSDATE()`** to be an alias for **`NOW()`**. The non-deterministic nature of **`SYSDATE()`** also means that indexes cannot be used for evaluating expressions that refer to it.

- **`TIME(expr)`**

Extracts the time part of the time or datetime expression *expr* and returns it as a string.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
mysql> -> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
mysql> -> '01:02:03.000123'
```

- **`TIMEDIFF(expr1,expr2)`**

`TIMEDIFF()` returns *expr1* – *expr2* expressed as a time value. *expr1* and *expr2* are time or date-and-time expressions, but both must be of the same type.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
mysql> -> '2000:01:01 00:00:00.000001');
mysql> -> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',
mysql> -> '1997-12-30 01:01:01.000002');
mysql> -> '46:58:57.999999'
```

- **`TIMESTAMP(expr)`, `TIMESTAMP(expr1,expr2)`**

With a single argument, this function returns the date or datetime expression *expr* as a datetime value. With two arguments, it adds the time expression *expr2* to the date or datetime expression *expr1* and returns the result as a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
mysql> -> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
mysql> -> '2004-01-01 00:00:00'
```

- **`TIMESTAMPADD(unit,interval,datetime_expr)`**

Adds the integer expression *interval* to the date or datetime expression *datetime_expr*. The unit for *interval* is given by the *unit* argument, which should be one of the following values: **`FRAC_SECOND`**, **`SECOND`**, **`MINUTE`**, **`HOURL`**, **`DAY`**, **`WEEK`**,

, `QUARTER`, or `YEAR`.

The *unit* value may be specified using one of keywords as shown, or with a prefix of `SQL_TSI_`. For example, `DAY` and `SQL_TSI_DAY` both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
-> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
-> '2003-01-09'
```

- `TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)`

Returns the integer difference between the date or datetime expressions *datetime_expr1* and *datetime_expr2*. The unit for the result is given by the *unit* argument. The legal values for *unit* are the same as those listed in the description of the `TIMESTAMPADD()` function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
-> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
-> -1
```

- `TIME_FORMAT(time,format)`

This is used like the `DATE_FORMAT()` function, but the *format* string may contain format specifiers only for hours, minutes, and seconds. Other specifiers produce a `NULL` value or `0`.

If the *time* value contains an hour part that is greater than 23, the `%H` and `%k` hour format specifiers produce a value larger than the usual range of 0..23. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00','%H %k %h %I %l');
-> '100 100 04 04 4'
```

- `TIME_TO_SEC(time)`

Returns the *time* argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
-> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
-> 2378
```

- `TO_DAYS(date)`

Given a date *date*, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
-> 728779
mysql> SELECT TO_DAYS('1997-10-07');
-> 729669
```

`TO_DAYS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See [Section 11.7, “What Calendar Is Used By MySQL?”](#), for details.

Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in [Section 10.3, “Date and Time Types”](#). For example, `'1997-10-07'` and `'97-10-07'` are seen as identical dates:

```
mysql> SELECT TO_DAYS('1997-10-07'), TO_DAYS('97-10-07');
-> 729669, 729669
```

- `UNIX_TIMESTAMP()`, `UNIX_TIMESTAMP(date)`

If called with no argument, returns a Unix timestamp (seconds since `'1970-01-01 00:00:00'` UTC) as an unsigned integer. If `UNIX_TIMESTAMP()` is called with a *date* argument, it returns the value of the argument as seconds since `'1970-01-01 00:00:00'` UTC. *date* may be a `DATE` string, a `DATETIME` string, a `TIMESTAMP`, or a number in the format `YYMMDD` or `YYYYMMDD`. The server interprets *date* as a value in the current time zone and converts it to an internal value in UTC. Clients can set their time zone as described in [Section 5.10.8, “MySQL Server Time Zone Support”](#).

```
mysql> SELECT UNIX_TIMESTAMP();
-> 882226357
mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
-> 875996580
```

When `UNIX_TIMESTAMP` is used on a `TIMESTAMP` column, the function returns the internal timestamp value directly, with no implicit “string-to-Unix-timestamp” conversion. If you pass an out-of-range date to `UNIX_TIMESTAMP()`, it returns `0`.

Note: If you use `UNIX_TIMESTAMP()` and `FROM_UNIXTIME()` to convert between `TIMESTAMP` values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For example, due to conventions for local time zone changes, it is possible for two `UNIX_TIMESTAMP()` to map two `TIMESTAMP` values to the same Unix timestamp value. `FROM_UNIXTIME()` will map that value back to only one of the original `TIMESTAMP` values. Here is an example, using `TIMESTAMP` values in the CET time zone:

```
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+-----+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+-----+
| 1111885200 |
+-----+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+-----+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+-----+
| 1111885200 |
+-----+
mysql> SELECT FROM_UNIXTIME(1111885200);
+-----+
| FROM_UNIXTIME(1111885200) |
+-----+
| 2005-03-27 03:00:00 |
+-----+
```

If you want to subtract `UNIX_TIMESTAMP()` columns, you might want to cast the result to signed integers. See [Section 11.9, “Cast Functions and Operators”](#).

- `UTC_DATE, UTC_DATE()`

Returns the current UTC date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
-> '2003-08-14', 20030814
```

- `UTC_TIME, UTC_TIME()`

Returns the current UTC time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
-> '18:07:53', 180753
```

- `UTC_TIMESTAMP, UTC_TIMESTAMP()`

Returns the current UTC date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
-> '2003-08-14 18:08:04', 20030814180804
```

- `WEEK(date[,mode])`

This function returns the week number for *date*. The two-argument form of `WEEK()` allows you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from `0` to `53` or from `1` to `53`. If the *mode* argument is omitted, the value of the `default_week_format` system variable is used. See [Section 5.2.3, “System Variables”](#).

The following table describes how the *mode* argument works.

	First day		

Mode	of week	Range	Week 1 is the first week ...
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with more than 3 days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with more than 3 days this year
4	Sunday	0-53	with more than 3 days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with more than 3 days this year
7	Monday	1-53	with a Monday in this year

```
mysql> SELECT WEEK('1998-02-20');
-> 7
mysql> SELECT WEEK('1998-02-20',0);
-> 7
mysql> SELECT WEEK('1998-02-20',1);
-> 8
mysql> SELECT WEEK('1998-12-31',1);
-> 53
```

Note that if a date falls in the last week of the previous year, MySQL returns 0 if you do not use 2, 3, 6, or 7 as the optional *mode* argument:

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
-> 2000, 0
```

One might argue that MySQL should return 52 for the `WEEK()` function, because the given date actually occurs in the 52nd week of 1999. We decided to return 0 instead because we want the function to return “the week number in the given year.” This makes use of the `WEEK()` function reliable when combined with other functions that extract a date part from a date.

If you would prefer the result to be evaluated with respect to the year that contains the first day of the week for the given date, use 0, 2, 5, or 7 as the optional *mode* argument.

```
mysql> SELECT WEEK('2000-01-01',2);
-> 52
```

Alternatively, use the `YEARWEEK()` function:

```
mysql> SELECT YEARWEEK('2000-01-01');
-> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
-> '52'
```

- `WEEKDAY(date)`

Returns the weekday index for *date* (0 = Monday, 1 = Tuesday, ... 6 = Sunday).

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');
-> 1
mysql> SELECT WEEKDAY('1997-11-05');
-> 2
```

- `WEEKOFYEAR(date)`

Returns the calendar week of the date as a number in the range from 1 to 53. `WEEKOFYEAR()` is a compatibility function that is equivalent to `WEEK(date,3)`.

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
-> 8
```

- `YEAR(date)`

Returns the year for *date*, in the range 1000 to 9999, or 0 for the “zero” date.

```
mysql> SELECT YEAR('98-02-03');
-> 1998
```

- **YEARWEEK(*date*), YEARWEEK(*date*,*mode*)**

Returns year and week for a date. The *mode* argument works exactly like the *mode* argument to **WEEK()**. The year in the result may be different from the year in the date argument for the first and the last week of the year.

```
mysql> SELECT YEARWEEK('1987-01-01');
-> 198653
```

Note that the week number is different from what the **WEEK()** function would return (0) for optional arguments 0 or 1, as **WEEK()** then returns the week in the context of the given year.

11.7. What Calendar Is Used By MySQL?

MySQL uses what is known as a *proleptic Gregorian calendar*.

Every country that has switched from the Julian to the Gregorian calendar has had to discard at least ten days during the switch. To see how this works, consider the month of October 1582, when the first Julian-to-Gregorian switch occurred:

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

There are no dates between October 4 and October 15. This discontinuity is called the *cutover*. Any dates before the cutover are Julian, and any dates following the cutover are Gregorian. Dates during a cutover are non-existent.

A calendar applied to dates when it wasn't actually in use is called *proleptic*. Thus, if we assume there was never a cutover and Gregorian rules always rule, we have a proleptic Gregorian calendar. This is what is used by MySQL, as is required by standard SQL. For this reason, dates prior to the cutover stored as MySQL **DATE** or **DATETIME** values must be adjusted to compensate for the difference. It is important to realize that the cutover did not occur at the same time in all countries, and that the later it happened, the more days were lost. For example, in Great Britain, it took place in 1752, when Wednesday September 2 was followed by Thursday September 14. Russia remained on the Julian calendar until 1918, losing 13 days in the process, and what is popularly referred to as its “October Revolution” occurred in November according to the Gregorian calendar.

11.8. Full-Text Search Functions

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])
search_modifier:
{
  IN BOOLEAN MODE
  | IN NATURAL LANGUAGE MODE
  | IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
  | WITH QUERY EXPANSION
}
```

MySQL has support for full-text indexing and searching:

- A full-text index in MySQL is an index of type **FULLTEXT**.
- Full-text indexes can be used only with **MyISAM** tables, and can be created only for **CHAR**, **VARCHAR**, or **TEXT** columns.
- A **FULLTEXT** index definition can be given in the **CREATE TABLE** statement when a table is created, or added later using **ALTER TABLE** or **CREATE INDEX**.

- For large datasets, it is much faster to load your data into a table that has no `FULLTEXT` index and then create the index after that, than to load data into a table that has an existing `FULLTEXT` index.

Full-text searching is performed using `MATCH() ... AGAINST` syntax. `MATCH()` takes a comma-separated list that names the columns to be searched. `AGAINST` takes a string to search for, and an optional modifier that indicates what type of search to perform. The search string must be a literal string, not a variable or a column name. There are three types of full-text searches:

- A boolean search interprets the search string using the rules of a special query language. The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual. Common words such as “some” or “then” are stopwords and do not match if present in the search string. The `IN BOOLEAN MODE` modifier specifies a boolean search. For more information, see [Section 11.8.1, “Boolean Full-Text Searches”](#).
- A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators. The stopword list applies. In addition, words that are present in more than 50% of the rows are considered common and do not match. Full-text searches are natural language searches if the `IN NATURAL LANGUAGE MODE` modifier is given or if no modifier is given.
- A query expansion search is a modification of a natural language search. The search string is used to perform a natural language search. Then words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search. The `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` or `WITH QUERY EXPANSION` modifier specifies a query expansion search. For more information, see [Section 11.8.2, “Full-Text Searches with Query Expansion”](#).

The `IN NATURAL LANGUAGE MODE` and `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` modifiers were added in MySQL 5.1.7.

Constraints on full-text searching are listed in [Section 11.8.4, “Full-Text Restrictions”](#).

```
mysql> CREATE TABLE articles (
->   id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
->   title VARCHAR(200),
->   body TEXT,
->   FULLTEXT (title,body)
-> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO articles (title,body) VALUES
-> ('MySQL Tutorial','DBMS stands for DataBase ...'),
-> ('How To Use MySQL Well','After you went through a ...'),
-> ('Optimizing MySQL','In this tutorial we will show ...'),
-> ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
-> ('MySQL vs. YourSQL','In the following database comparison ...'),
-> ('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' IN NATURAL LANGUAGE MODE);
+----+-----+-----+
| id | title                | body                |
+----+-----+-----+
| 5  | MySQL vs. YourSQL    | In the following database comparison ... |
| 1  | MySQL Tutorial       | DBMS stands for DataBase ... |
+----+-----+-----+
2 rows in set (0.00 sec)
```

The `MATCH()` function performs a natural language search for a string against a *text collection*. A collection is a set of one or more columns included in a `FULLTEXT` index. The search string is given as the argument to `AGAINST()`. For each row in the table, `MATCH()` returns a relevance value; that is, a similarity measure between the search string and the text in that row in the columns named in the `MATCH()` list.

By default, the search is performed in case-insensitive fashion. However, you can perform a case-sensitive full-text search by using a binary collation for the indexed columns. For example, a column that uses the `latin1` character set of can be assigned a collation of `latin1_bin` to make it case sensitive for full-text searches.

When `MATCH()` is used in a `WHERE` clause, as in the example shown earlier, the rows returned are automatically sorted with the highest relevance first. Relevance values are non-negative floating-point numbers. Zero relevance means no similarity. Relevance is

computed based on the number of words in the row, the number of unique words in that row, the total number of words in the collection, and the number of documents (rows) that contain a particular word.

To simply count matches, you could use a query like this:

```
mysql> SELECT COUNT(*) FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+
| COUNT(*) |
+-----+
|          2 |
+-----+
1 row in set (0.00 sec)
```

However, you might find it quicker to rewrite the query as follows:

```
mysql> SELECT
-> COUNT(IF(MATCH (title,body) AGAINST ('database' IN NATURAL LANGUAGE MODE), 1, NULL))
-> AS count
-> FROM articles;
+-----+
| count |
+-----+
|          2 |
+-----+
1 row in set (0.03 sec)
```

The first query sorts the results by relevance whereas the second does not. However, the second query performs a full table scan and the first does not. The first may be faster if the search matches few rows; otherwise, the second may be faster because it would read many rows anyway.

For natural-language full-text searches, it is a requirement that the columns named in the `MATCH()` function be the same columns included in some `FULLTEXT` index in your table. For the preceding query, note that the columns named in the `MATCH()` function (`title` and `body`) are the same as those named in the definition of the `article` table's `FULLTEXT` index. If you wanted to search the `title` or `body` separately, you would need to create separate `FULLTEXT` indexes for each column.

It is also possible to perform a boolean search or a search with query expansion. These search types are described in [Section 11.8.1, “Boolean Full-Text Searches”](#), and [Section 11.8.2, “Full-Text Searches with Query Expansion”](#).

A full-text search that uses an index can name columns only from a single table in the `MATCH()` clause because an index cannot span multiple tables. A boolean search can be done in the absence of an index (albeit more slowly), in which case it is possible to name columns from multiple tables.

The preceding example is a basic illustration that shows how to use the `MATCH()` function where rows are returned in order of decreasing relevance. The next example shows how to retrieve the relevance values explicitly. Returned rows are not ordered because the `SELECT` statement includes neither `WHERE` nor `ORDER BY` clauses:

```
mysql> SELECT id, MATCH (title,body)
-> AGAINST ('Tutorial' IN NATURAL LANGUAGE MODE) AS score
-> FROM articles;
+-----+
| id | score |
+-----+
| 1 | 0.65545833110809 |
| 2 | 0 |
| 3 | 0.66266459226608 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
+-----+
6 rows in set (0.00 sec)
```

The following example is more complex. The query returns the relevance values and it also sorts the rows in order of decreasing relevance. To achieve this result, you should specify `MATCH()` twice: once in the `SELECT` list and once in the `WHERE` clause. This causes no additional overhead, because the MySQL optimizer notices that the two `MATCH()` calls are identical and invokes the full-text search code only once.

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root'
-> IN NATURAL LANGUAGE MODE) AS score
-> FROM articles WHERE MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root'
-> IN NATURAL LANGUAGE MODE);
+-----+-----+-----+
| id | body | MATCH (title,body) AGAINST
```

id	body	score
4	1. Never run mysqld as root. 2. ...	1.5219271183014
6	When configured properly, MySQL ...	1.3114095926285

2 rows in set (0.00 sec)

The MySQL **FULLTEXT** implementation regards any sequence of true word characters (letters, digits, and underscores) as a word. That sequence may also contain apostrophes (''), but not more than one in a row. This means that `aaa'bbb` is regarded as one word, but `aaa'bbb` is regarded as two words. Apostrophes at the beginning or the end of a word are stripped by the **FULLTEXT** parser; `'aaa'bbb` would be parsed as `aaa'bbb`.

The **FULLTEXT** parser determines where words start and end by looking for certain delimiter characters; for example, ' ' (space), ',', (comma), and '.' (period). If words are not separated by delimiters (as in, for example, Chinese), the **FULLTEXT** parser cannot determine where a word begins or ends. To be able to add words or other indexed terms in such languages to a **FULLTEXT** index, you must preprocess them so that they are separated by some arbitrary delimiter such as ''.

In MySQL 5.1, it is possible to write a plugin that replaces the built-in full-text parser. For details, see [Section 28.2, “The MySQL Plugin Interface”](#). For example parser plugin source code, see the [plugin/fulltext](#) directory of a MySQL source distribution.

Some words are ignored in full-text searches:

- Any word that is too short is ignored. The default minimum length of words that are found by full-text searches is four characters.
- Words in the stopword list are ignored. A stopword is a word such as “the” or “some” that is so common that it is considered to have zero semantic value. There is a built-in stopword list, but it can be overwritten by a user-defined list.

The default stopword list is given in [Section 11.8.3, “Full-Text Stopwords”](#). The default minimum word length and stopword list can be changed as described in [Section 11.8.5, “Fine-Tuning MySQL Full-Text Search”](#).

Every correct word in the collection and in the query is weighted according to its significance in the collection or query. Consequently, a word that is present in many documents has a lower weight (and may even have a zero weight), because it has lower semantic value in this particular collection. Conversely, if the word is rare, it receives a higher weight. The weights of the words are combined to compute the relevance of the row.

Such a technique works best with large collections (in fact, it was carefully tuned this way). For very small tables, word distribution does not adequately reflect their semantic value, and this model may sometimes produce bizarre results. For example, although the word “MySQL” is present in every row of the `articles` table shown earlier, a search for the word produces no results:

```
mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('MySQL' IN NATURAL LANGUAGE MODE);
Empty set (0.00 sec)
```

The search result is empty because the word “MySQL” is present in at least 50% of the rows. As such, it is effectively treated as a stopword. For large datasets, this is the most desirable behavior: A natural language query should not return every second row from a 1GB table. For small datasets, it may be less desirable.

A word that matches half of the rows in a table is less likely to locate relevant documents. In fact, it most likely finds plenty of irrelevant documents. We all know this happens far too often when we are trying to find something on the Internet with a search engine. It is with this reasoning that rows containing the word are assigned a low semantic value for *the particular dataset in which they occur*. A given word may exceed the 50% threshold in one dataset but not another.

The 50% threshold has a significant implication when you first try full-text searching to see how it works: If you create a table and insert only one or two rows of text into it, every word in the text occurs in at least 50% of the rows. As a result, no search returns any results. Be sure to insert at least three rows, and preferably many more. Users who need to bypass the 50% limitation can use the boolean search mode; see [Section 11.8.1, “Boolean Full-Text Searches”](#).

11.8.1. Boolean Full-Text Searches

MySQL can perform boolean full-text searches using the `IN BOOLEAN MODE` modifier:

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
-> AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+-----+-----+
| id | title | body |
+-----+-----+
+-----+-----+
+-----+-----+
```

1	MySQL Tutorial	DBMS stands for DataBase ...
2	How To Use MySQL Well	After you went through a ...
3	Optimizing MySQL	In this tutorial we will show ...
4	1001 MySQL Tricks	1. Never run mysqld as root. 2. ...
6	MySQL Security	When configured properly, MySQL ...

The `+` and `-` operators indicate that a word is required to be present or absent, respectively, for a match to occur. Thus, this query retrieves all the rows that contain the word “MySQL” but that do *not* contain the word “YourSQL”.

Boolean full-text searches have these characteristics:

- They do not use the 50% threshold.
- They do not automatically sort rows in order of decreasing relevance. You can see this from the preceding query result: The row with the highest relevance is the one that contains “MySQL” twice, but it is listed last, not first.
- They can work even without a `FULLTEXT` index, although a search executed in this fashion would be quite slow.
- The minimum and maximum word length full-text parameters apply.
- The stopword list applies.

The boolean full-text search capability supports the following operators:

- `+`

A leading plus sign indicates that this word *must* be present in each row that is returned.

- `-`

A leading minus sign indicates that this word must *not* be present in any of the rows that are returned.

Note: The `-` operator acts only to exclude rows that are otherwise matched by other search terms. Thus, a boolean-mode search that contains only terms preceded by `-` returns an empty result. It does not return “all rows except those containing any of the excluded terms.”

- (no operator)

By default (when neither `+` nor `-` is specified) the word is optional, but the rows that contain it are rated higher. This mimics the behavior of `MATCH() ... AGAINST()` without the `IN BOOLEAN MODE` modifier.

- `>` `<`

These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The `>` operator increases the contribution and the `<` operator decreases it. See the example following this list.

- `( )`

Parentheses group words into subexpressions. Parenthesized groups can be nested.

- `~`

A leading tilde acts as a negation operator, causing the word's contribution to the row's relevance to be negative. This is useful for marking “noise” words. A row containing such a word is rated lower than others, but is not excluded altogether, as it would be with the `-` operator.

- `*`

The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it should be *appended* to the word to be affected. Words match if they begin with the word preceding the `*` operator.

If a stopword or too-short word is specified with the truncation operator, it will not be stripped from a boolean query. For example, a search for `'+word +stopword*'` will likely return fewer rows than a search for `'+word +stopword'` because the former query remains as is and requires `stopword*` to be present in a document. The latter query is transformed to `+word`.

- `"`

A phrase that is enclosed within double quote (`"`) characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words, performs a search in the `FULLTEXT` index for the words. Non-word characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, `"test phrase"` matches `"test, phrase"`.

If the phrase contains no words that are in the index, the result is empty. For example, if all words are either stopwords or shorter than the minimum length of indexed words, the result is empty.

The following examples demonstrate some search strings that use boolean full-text operators:

- `'apple banana'`

Find rows that contain at least one of the two words.

- `'+apple +juice'`

Find rows that contain both words.

- `'+apple macintosh'`

Find rows that contain the word “apple”, but rank rows higher if they also contain “macintosh”.

- `'+apple -macintosh'`

Find rows that contain the word “apple” but not “macintosh”.

- `'+apple ~macintosh'`

Find rows that contain the word “apple”, but if the row also contains the word “macintosh”, rate it lower than if row does not. This is “softer” than a search for `'+apple -macintosh'`, for which the presence of “macintosh” causes the row not to be returned at all.

- `'+apple +(>turnover <strudel)'`

Find rows that contain the words “apple” and “turnover”, or “apple” and “strudel” (in any order), but rank “apple turnover” higher than “apple strudel”.

- `'apple*'`

Find rows that contain words such as “apple”, “apples”, “applesauce”, or “applet”.

- `'"some words"'`

Find rows that contain the exact phrase “some words” (for example, rows that contain “some words of wisdom” but not “some noise words”). Note that the `"` characters that enclose the phrase are operator characters that delimit the phrase. They are not the quotes that enclose the search string itself.

11.8.2. Full-Text Searches with Query Expansion

Full-text search supports query expansion (and in particular, its variant “blind query expansion”). This is generally useful when a search phrase is too short, which often means that the user is relying on implied knowledge that the full-text search engine lacks. For example, a user searching for “database” may really mean that “MySQL”, “Oracle”, “DB2”, and “RDBMS” all are phrases that should match “databases” and should be returned, too. This is implied knowledge.

Blind query expansion (also known as automatic relevance feedback) is enabled by adding `WITH QUERY EXPANSION` or `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` following the search phrase. It works by performing the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. Thus, if one of these documents contains the word “databases” and the word “MySQL”, the second search finds the documents that contain the word “MySQL” even if they do not contain the word “database”. The following example shows this difference:

```
mysql> SELECT * FROM articles
```

```

-> WHERE MATCH (title,body)
-> AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+-----+-----+
| id | title | body |
+-----+-----+-----+
| 5 | MySQL vs. YourSQL | In the following database comparison ... |
| 1 | MySQL Tutorial | DBMS stands for DataBase ... |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' WITH QUERY EXPANSION);
+-----+-----+-----+
| id | title | body |
+-----+-----+-----+
| 1 | MySQL Tutorial | DBMS stands for DataBase ... |
| 5 | MySQL vs. YourSQL | In the following database comparison ... |
| 3 | Optimizing MySQL | In this tutorial we will show ... |
+-----+-----+-----+
3 rows in set (0.00 sec)

```

Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell “Maigret”. A search for “Megre and the reluctant witnesses” finds only “Maigret and the Reluctant Witnesses” without query expansion. A search with query expansion finds all books with the word “Maigret” on the second pass.

Note

Because blind query expansion tends to increase noise significantly by returning non-relevant documents, it is meaningful to use only when a search phrase is rather short.

11.8.3. Full-Text Stopwords

The following table shows the default list of full-text stopwords.

a's	able	about	above	according
accordingly	across	actually	after	afterwards
again	against	ain't	all	allow
allows	almost	alone	along	already
also	although	always	am	among
amongst	an	and	another	any
anybody	anyhow	anyone	anything	anyway
anyways	anywhere	apart	appear	appreciate
appropriate	are	aren't	around	as
aside	ask	asking	associated	at
available	away	awfully	be	became
because	become	becomes	becoming	been
before	beforehand	behind	being	believe
below	beside	besides	best	better
between	beyond	both	brief	but
by	c'mon	c's	came	can
can't	cannot	cant	cause	causes
certain	certainly	changes	clearly	co
com	come	comes	concerning	consequently
consider	considering	contain	containing	contains
corresponding	could	couldn't	course	currently
definitely	described	despite	did	didn't
different	do	does	doesn't	doing
don't	done	down	downwards	during

each	edu	eg	eight	either
else	elsewhere	enough	entirely	especially
et	etc	even	ever	every
everybody	everyone	everything	everywhere	ex
exactly	example	except	far	few
fifth	first	five	followed	following
follows	for	former	formerly	forth
four	from	further	furthermore	get
gets	getting	given	gives	go
goes	going	gone	got	gotten
greetings	had	hadn't	happens	hardly
has	hasn't	have	haven't	having
he	he's	hello	help	hence
her	here	here's	hereafter	hereby
herein	hereupon	hers	herself	hi
him	himself	his	hither	hopefully
how	howbeit	however	i'd	i'll
i'm	i've	ie	if	ignored
immediate	in	inasmuch	inc	indeed
indicate	indicated	indicates	inner	insofar
instead	into	inward	is	isn't
it	it'd	it'll	it's	its
itself	just	keep	keeps	kept
know	knows	known	last	lately
later	latter	latterly	least	less
lest	let	let's	like	liked
likely	little	look	looking	looks
ltd	mainly	many	may	maybe
me	mean	meanwhile	merely	might
more	moreover	most	mostly	much
must	my	myself	name	namely
nd	near	nearly	necessary	need
needs	neither	never	nevertheless	new
next	nine	no	nobody	non
none	noone	nor	normally	not
nothing	novel	now	nowhere	obviously
of	off	often	oh	ok
okay	old	on	once	one
ones	only	onto	or	other
others	otherwise	ought	our	ours
ourselves	out	outside	over	overall
own	particular	particularly	per	perhaps
placed	please	plus	possible	presumably
probably	provides	que	quite	qv
rather	rd	re	really	reasonably

regarding	regardless	regards	relatively	respectively
right	said	same	saw	say
saying	says	second	secondly	see
seeing	seem	seemed	seeming	seems
seen	self	selves	sensible	sent
serious	seriously	seven	several	shall
she	should	shouldn't	since	six
so	some	somebody	somehow	someone
something	sometime	sometimes	somewhat	somewhere
soon	sorry	specified	specify	specifying
still	sub	such	sup	sure
t's	take	taken	tell	tends
th	than	thank	thanks	thanx
that	that's	thats	the	their
theirs	them	themselves	then	thence
there	there's	thereafter	thereby	therefore
therein	theres	thereupon	these	they
they'd	they'll	they're	they've	think
third	this	thorough	thoroughly	those
though	three	through	throughout	thru
thus	to	together	too	took
toward	towards	tried	tries	truly
try	trying	twice	two	un
under	unfortunately	unless	unlikely	until
unto	up	upon	us	use
used	useful	uses	using	usually
value	various	very	via	viz
vs	want	wants	was	wasn't
way	we	we'd	we'll	we're
we've	welcome	well	went	were
weren't	what	what's	whatever	when
whence	whenever	where	where's	whereafter
whereas	whereby	wherein	whereupon	wherever
whether	which	while	whither	who
who's	whoever	whole	whom	whose
why	will	willing	wish	with
within	without	won't	wonder	would
would	wouldn't	yes	yet	you
you'd	you'll	you're	you've	your
yours	yourself	yourselves	zero	

11.8.4. Full-Text Restrictions

- Full-text searches are supported for [MyISAM](#) tables only.

- Full-text searches can be used with most multi-byte character sets. The exception is that for Unicode, the `utf8` character set can be used, but not the `ucs2` character set.
- Ideographic languages such as Chinese and Japanese do not have word delimiters. Therefore, the `FULLTEXT` parser *cannot determine where words begin and end in these and other such languages*. The implications of this and some workarounds for the problem are described in [Section 11.8, “Full-Text Search Functions”](#).
- Although the use of multiple character sets within a single table is supported, all columns in a `FULLTEXT` index must use the same character set and collation.
- The `MATCH()` column list must match exactly the column list in some `FULLTEXT` index definition for the table, unless this `MATCH()` is `IN BOOLEAN MODE`. Boolean-mode searches can be done on non-indexed columns, although they are likely to be slow.
- The argument to `AGAINST()` must be a constant string.

11.8.5. Fine-Tuning MySQL Full-Text Search

MySQL's full-text search capability has few user-tunable parameters. You can exert more control over full-text searching behavior if you have a MySQL source distribution because some changes require source code modifications. See [Section 2.9, “MySQL Installation Using a Source Distribution”](#).

Note that full-text search is carefully tuned for the most effectiveness. Modifying the default behavior in most cases can actually decrease effectiveness. *Do not alter the MySQL sources unless you know what you are doing.*

Most full-text variables described in this section must be set at server startup time. A server restart is required to change them; they cannot be modified while the server is running.

Some variable changes require that you rebuild the `FULLTEXT` indexes in your tables. Instructions for doing this are given at the end of this section.

- The minimum and maximum lengths of words to be indexed are defined by the `ft_min_word_len` and `ft_max_word_len` system variables. (See [Section 5.2.3, “System Variables”](#).) The default minimum value is four characters; the default maximum is version dependent. If you change either value, you must rebuild your `FULLTEXT` indexes. For example, if you want three-character words to be searchable, you can set the `ft_min_word_len` variable by putting the following lines in an option file:

```
[mysqld]
ft_min_word_len=3
```

Then you must restart the server and rebuild your `FULLTEXT` indexes. Note particularly the remarks regarding `myisamchk` in the instructions following this list.

- To override the default stopword list, set the `ft_stopword_file` system variable. (See [Section 5.2.3, “System Variables”](#).) The variable value should be the pathname of the file containing the stopword list, or the empty string to disable stopword filtering. After changing the value of this variable or the contents of the stopword file, restart the server and rebuild your `FULLTEXT` indexes.

The stopword list is free-form. That is, you may use any non-alphanumeric character such as newline, space, or comma to separate stopwords. Exceptions are the underscore character (`'_'`) and a single apostrophe (`' '`) which are treated as part of a word. The character set of the stopword list is the server's default character set; see [Section 9.3.1, “Server Character Set and Collation”](#).

- The 50% threshold for natural language searches is determined by the particular weighting scheme chosen. To disable it, look for the following line in `storage/myisam/ftdefs.h`:

```
#define GWS_IN_USE GWS_PROB
```

Change that line to this:

```
#define GWS_IN_USE GWS_FREQ
```

Then recompile MySQL. There is no need to rebuild the indexes in this case. **Note:** By making this change, you *severely* decrease MySQL's ability to provide adequate relevance values for the `MATCH()` function. If you really need to search for such common words, it would be better to search using `IN BOOLEAN MODE` instead, which does not observe the 50% threshold.

- To change the operators used for boolean full-text searches, set the `ft_boolean_syntax` system variable. This variable can be changed while the server is running, but you must have the `SUPER` privilege to do so. No rebuilding of indexes is necessary in this case. See [Section 5.2.3, “System Variables”](#), which describes the rules governing how to set this variable.
- If you want to change the set of characters that are considered word characters, you can do so in two ways. Suppose that you want to treat the hyphen character ('-') as a word character. Use either of these methods:
 - Modify the MySQL source: In `storage/myisam/ftdefs.h`, see the `true_word_char()` and `misc_word_char()` macros. Add '-' to one of those macros and recompile MySQL.
 - Modify a character set file: This requires no recompilation. The `true_word_char()` macro uses a “character type” table to distinguish letters and numbers from other characters. You can edit the `<ctype><map>` contents in one of the character set XML files to specify that '-' is a “letter.” Then use the given character set for your `FULLTEXT` indexes.

After making the modification, you must rebuild the indexes for each table that contains any `FULLTEXT` indexes.

If you modify full-text variables that affect indexing (`ft_min_word_len`, `ft_max_word_len`, or `ft_stopword_file`), or if you change the stopwords file itself, you must rebuild your `FULLTEXT` indexes after making the changes and restarting the server. To rebuild the indexes in this case, it is sufficient to do a `QUICK` repair operation:

```
mysql> REPAIR TABLE tbl_name QUICK;
```

Each table that contains any `FULLTEXT` index must be repaired as just shown. Otherwise, queries for the table may yield incorrect results, and modifications to the table will cause the server to see the table as corrupt and in need of repair.

Note that if you use `myisamchk` to perform an operation that modifies table indexes (such as repair or analyze), the `FULLTEXT` indexes are rebuilt using the *default* full-text parameter values for minimum word length, maximum word length, and stopwords file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in `MyISAM` index files. To avoid the problem if you have modified the minimum or maximum word length or stopwords file values used by the server, specify the same `ft_min_word_len`, `ft_max_word_len`, and `ft_stopword_file` values to `myisamchk` that you use for `mysqld`. For example, if you have set the minimum word length to 3, you can repair a table with `myisamchk` like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that `myisamchk` and the server use the same values for full-text parameters, place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using `myisamchk` is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE` statements. These statements are performed by the server, which knows the proper full-text parameter values to use.

11.9. Cast Functions and Operators

Name	Description
<code>BINARY</code>	Cast a string to a binary string
<code>CAST()</code>	Cast a value as a certain type

- `BINARY`

The `BINARY` operator casts the string following it to a binary string. This is an easy way to force a column comparison to be done byte by byte rather than character by character. This causes the comparison to be case sensitive even if the column isn't defined as `BINARY` or `BLOB`. `BINARY` also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
```

```
mysql> -> 1
mysql> SELECT BINARY 'a' = 'A';
mysql> -> 0
mysql> SELECT 'a' = 'a ';
mysql> -> 1
mysql> SELECT BINARY 'a' = 'a ';
mysql> -> 0
```

In a comparison, **BINARY** affects the entire operation; it can be given before either operand with the same result.

BINARY *str* is shorthand for **CAST(str AS BINARY)**.

Note that in some contexts, if you cast an indexed column to **BINARY**, MySQL is not able to use the index efficiently.

- **CAST(expr AS type)**, **CONVERT(expr, type)**, **CONVERT(expr USING transcoding_name)**

The **CAST()** and **CONVERT()** functions take a value of one type and produce a value of another type.

The *type* can be one of the following values:

- **BINARY[(N)]**
- **CHAR[(N)]**
- **DATE**
- **DATETIME**
- **DECIMAL**
- **SIGNED [INTEGER]**
- **TIME**
- **UNSIGNED [INTEGER]**

BINARY produces a string with the **BINARY** data type. See [Section 10.4.2, “The BINARY and VARBINARY Types”](#) for a description of how this affects comparisons. If the optional length *N* is given, **BINARY(N)** causes the cast to use no more than *N* bytes of the argument. Values shorter than *N* bytes are padded with **0x00** bytes to a length of *N*.

CHAR(N) causes the cast to use no more than *N* characters of the argument.

CAST() and **CONVERT(... USING ...)** are standard SQL syntax. The non-**USING** form of **CONVERT()** is ODBC syntax.

CONVERT() with **USING** is used to convert data between different character sets. In MySQL, transcoding names are the same as the corresponding character set names. For example, this statement converts the string **'abc'** in the default character set to the corresponding string in the **utf8** character set:

```
SELECT CONVERT('abc' USING utf8);
```

Normally, you cannot compare a **BLOB** value or other binary string in case-insensitive fashion because binary strings have no character set, and thus no concept of lettercase. To perform a case-insensitive comparison, use the **CONVERT()** function to convert the value to a non-binary string. If the character set of the result has a case-insensitive collation, the **LIKE** operation is not case sensitive:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) FROM tbl_name;
```

To use a different character set, substitute its name for **latin1** in the preceding statement. To ensure that a case-insensitive collation is used, specify a **COLLATE** clause following the **CONVERT()** call.

CONVERT() can be used more generally for comparing strings that are represented in different character sets.

The cast functions are useful when you want to create a column with a specific type in a **CREATE ... SELECT** statement:

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

The functions also can be useful for sorting [ENUM](#) columns in lexical order. Normally, sorting of [ENUM](#) columns occurs using the internal numeric values. Casting the values to [CHAR](#) results in a lexical sort:

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

`CAST(str AS BINARY)` is the same thing as `BINARY str`. `CAST(expr AS CHAR)` treats the expression as a string with the default character set.

`CAST()` also changes the result if you use it as part of a more complex expression such as `CONCAT('Date: ', CAST(NOW() AS DATE))`.

You should not use `CAST()` to extract data in different formats but instead use string functions like `LEFT()` or `EXTRACT()`. See [Section 11.6, “Date and Time Functions”](#).

To cast a string to a numeric value in numeric context, you normally do not have to do anything other than to use the string value as though it were a number:

```
mysql> SELECT 1+'1';
-> 2
```

If you use a number in string context, the number automatically is converted to a [BINARY](#) string.

```
mysql> SELECT CONCAT('hello you ',2);
-> 'hello you 2'
```

MySQL supports arithmetic with both signed and unsigned 64-bit values. If you are using numeric operators (such as `+` or `-`) and one of the operands is an unsigned integer, the result is unsigned. You can override this by using the [SIGNED](#) and [UNSIGNED](#) cast operators to cast the operation to a signed or unsigned 64-bit integer, respectively.

```
mysql> SELECT CAST(1-2 AS UNSIGNED)
-> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
-> -1
```

Note that if either operand is a floating-point value, the result is a floating-point value and is not affected by the preceding rule. (In this context, [DECIMAL](#) column values are regarded as floating-point values.)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
-> -1.0
```

If you are using a string in an arithmetic operation, this is converted to a floating-point number.

If you convert a “zero” date string to a date, `CONVERT()` and `CAST()` return [NULL](#) and produce a warning when the [NO_ZERO_DATE](#) SQL mode is enabled.

11.10. XML Functions

Name	Description
ExtractValue() (v5.1.5)	Extracts a value from an XML string using XPath notation
UpdateXML() (v5.1.5)	Return replaced XML fragment

This section discusses XML and related functionality in MySQL.

Note

It is possible to obtain XML-formatted output from MySQL in the `mysql` and `mysqldump` clients by invoking them with the `--xml` option. See [Section 7.8, “mysql — The MySQL Command-Line Tool”](#), and [Section 7.13, “mysqldump — A Database Backup Program”](#).

Beginning with MySQL 5.1.5, two functions providing basic XPath (XML Path Language) capabilities are available.

Note

These functions remain under development. We continue to improve these and other aspects of XML and XPath functionality in MySQL 5.1 and onwards. You may discuss these, ask questions about them, and obtain help from other users with them in the [MySQL XML User Forum](#).

Beginning with MySQL 5.1.20, user variables in the XPath expressions used with these functions are supported in two forms, whose notations differ according to whether they are weakly or strongly checked (see also [Bug#26518](#)):

- **Weak checking.** Variables using the syntax `$@variable_name` are not checked. No warnings or errors are issued by the server if a variable has the wrong type or has previously been assigned a value. This also means the user is fully responsible for any typographical errors, since no warnings will be given if (for example) `$@myvairable` is used where `$@myvariable` was intended.
- **Strong checking.** Variables using the syntax `$variable_name` can be declared and used with these functions when they are called inside stored procedures. Such variables are local to the stored procedure in which they are defined, and are strongly checked for type and value.

Expressions containing user-defined variables of either sort must otherwise (except for notation) conform to the rules for XPath expressions containing variables as given in the XPath specification.

- `ExtractValue(xml_frag, xpath_expr)`

`ExtractValue()` takes two string arguments, a fragment of XML markup `xml_frag` and an XPath expression `xpath_expr` (also known as a *locator*); it returns the text (CDATA) of the first text node which is a child of the element(s) matched by the XPath expression. It is the equivalent of performing a match using the `xpath_expr` after appending `/text()`. In other words, `ExtractValue('<a><b>Sakila</b></a>', '/a/b')` and `ExtractValue('<a><b>Sakila</b></a>', '/a/b/text()')` produce the same result.

If multiple matches are found, then the content of the first child text node of each matching element is returned (in the order matched) as a single, space-delimited string.

If no matching text node is found for the (augmented) expression — for whatever reason, as long as `xpth_expr` is valid, and `xml_frag` is well-formed — an empty string is returned. No distinction is made between a match on an empty element and no match at all. This is by design.

If you need to determine whether no matching element was found in `xml_frag` or such an element was found but contained no child text nodes, you should test the result of an expression that uses the XPath `count()` function. For example, both of these statements return an empty string, as shown here:

```
mysql> SELECT ExtractValue('<a><b></a>', '/a/b');
+-----+
| ExtractValue('<a><b></a>', '/a/b') |
+-----+
|                                     |
+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c></a>', '/a/b');
+-----+
| ExtractValue('<a><c></a>', '/a/b') |
+-----+
|                                     |
+-----+
1 row in set (0.00 sec)
```

However, you can determine whether there was actually a matching element using the following:

```
mysql> SELECT ExtractValue('<a><b></a>', 'count(/a/b)');
+-----+
| ExtractValue('<a><b></a>', 'count(/a/b)') |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c></a>', 'count(/a/b)');
+-----+
| ExtractValue('<a><c></a>', 'count(/a/b)') |
+-----+
| 0 |
+-----+
1 row in set (0.01 sec)
```

Important

`ExtractValue()` returns only `CDATA`, and does not return any tags that might be contained within a matching tag, nor any of their content (see the result returned as `val1` in the following example).

```
mysql> SELECT
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/a') AS val1,
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/a/b') AS val2,
-> ExtractValue('<a>ccc<b>ddd</b></a>', '//b') AS val3,
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/b') AS val4,
-> ExtractValue('<a>ccc<b>ddd</b><b>eee</b></a>', '//b') AS val5;

+-----+-----+-----+-----+-----+
| val1 | val2 | val3 | val4 | val5 |
+-----+-----+-----+-----+-----+
| ccc  | ddd  | ddd  |      | ddd eee |
+-----+-----+-----+-----+-----+
```

Beginning with MySQL 5.1.8, this function uses the current SQL collation for making comparisons with `contains()`. (Previously, binary — that is, case-sensitive — comparison was always used.)

Beginning with MySQL 5.1.12, `NULL` is returned if `xml_frag` is invalid XML, and a warning is generated, as shown in this example:

```
mysql> SELECT ExtractValue('<a>c</a><b>', '//a');
+-----+
| ExtractValue('<a>c</a><b>', '//a') |
+-----+
| NULL                               |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message
+-----+-----+-----+
| Warning | 1523 | Incorrect XML value: 'parse error at line 1 pos 11: END-OF-INPUT unexpected ('>' wanted)'
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
+-----+
| ExtractValue('<a>c</a><b/>', '//a') |
+-----+
| c                                   |
+-----+
1 row in set (0.00 sec)
```

Prior to MySQL 5.1.12, an empty string was returned in such cases. ([Bug#18201](#))

- `UpdateXML(xml_target, xpath_expr, new_xml)`

This function replaces a single portion of a given fragment of XML markup `xml_target` with a new XML fragment `new_xml`, and then returns the changed XML. The portion of `xml_target` that is replaced matches an XPath expression `xpath_expr` supplied by the user. If no expression matching `xpath_expr` is found, or if multiple matches are found, the function returns the original `xml_target` XML fragment. All three arguments must be strings.

```
mysql> SELECT
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>') AS val1,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val2,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '//b', '<e>fff</e>') AS val3,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val4,
-> UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val5
-> \G

***** 1. row *****
val1: <e>fff</e>
val2: <a><b>ccc</b><d></d></a>
val3: <a><e>fff</e><d></d></a>
val4: <a><b>ccc</b><e>fff</e><d></d></a>
val5: <a><d></d><b>ccc</b><d></d></a>
```

Descriptions and examples of some basic XPath expressions follow:

- `/tag`

Matches `<tag/>` if and only if `<tag/>` is the root element.

Example: `/a` has a match in `<a><b/></a>` because it matches the outermost (root) tag. It does not match the inner `a` element in `<b><a/></b>` because in this instance it is the child of another element.

- `/tag1/tag2`

Matches `<tag2/>` if and only if it is a child of `<tag1/>`, and `<tag1/>` is the root element.

Example: `/a/b` matches the `b` element in the XML fragment `<a><b/></a>` because it is a child of the root element `a`. It does not have a match in `<b><a/></b>` because in this case, `b` is the root element (and hence the child of no other element). Nor does the XPath expression have a match in `<a><c><b/></c></a>`; here, `b` is a descendant of `a`, but not actually a child of `a`.

This construct is extendable to three or more elements. For example, the XPath expression `/a/b/c` matches the `c` element in the fragment `<a><b><c/></b></a>`.

- `//tag`

Matches any instance of `<tag>`.

Example: `//a` matches the `a` element in any of the following: `<a><b><c/></b></a>`; `<c><a><b/></a></b>`; `<c><b><a/></b></c>`.

`//` can be combined with `/`. For example, `//a/b` matches the `b` element in either of the fragments `<a><b/></a>` or `<a><b><c/></b></a>`.

- The `*` operator acts as a “wildcard” that matches any element. For example, the expression `/*/b` matches the `b` element in either of the XML fragments `<a><b/></a>` or `<c><b/></c>`. However, the expression does not produce a match in the fragment `<b><a/></b>` because `b` must be a child of some other element. The wildcard may be used in any position: The expression `/*/*b/*` will match any child of a `b` element that is itself not the root element.
- Multiple locators may be matched using the `|` (logical OR) operator. For example, the expression `//b|//c` matches all `b` and `c` elements in the XML target.
- It is also possible to match an element based on the value of one or more of its attributes. This done using the syntax `tag[@attribute="value"]`. For example, the expression `//b[@id="idB"]` matches the second `b` element in the fragment `<a><b id="idA"/><c/><b id="idB"/></a>`. To match against any element having `attribute="value"`, use the XPath expression `//*[attribute="value"]`.

To filter multiple attribute values, simply use multiple attribute-comparison clauses in succession. For example, the expression `/b[@c="x"][@d="y"]` matches the element `<b c="x" d="y"/>` occurring anywhere in a given XML fragment.

To find elements for which the same attribute matches one of several values, you must use multiple locators joined by the `|` operator. For example, to match all `b` elements whose `c` attributes have either of the values 23 or 17, use the expression `/b[@c="23"]|b[@c="17"]`.

Note

A discussion in depth of XPath syntax and usage are beyond the scope of this Manual. Please see the [XML Path Language \(XPath\) 1.0 standard](#) for definitive information. A useful resource for those new to XPath or who are wishing a refresher in the basics is the [Zvon.org XPath Tutorial](#), which is available in several languages.

The XPath syntax supported by these functions is currently subject to the following limitations:

- Prior to MySQL 5.1.14, nodeset-to-nodeset comparison (such as `'/a/b[@c=@d]'`) was not supported, and equality and inequality (`=` and `!=`) were the only supported comparison operators. Beginning with MySQL 5.1.14, all of the standard XPath comparison operators are supported. ([Bug#22823](#))
- Relative locator expressions are not supported. XPath expressions must begin with `/` or `//`.
- The `::` operator is not supported.

- “Up-and-down” navigation is not supported in cases where the path would lead “above” the root element. That is, you cannot use expressions which match on descendants of ancestors of a given element, where one or more of the ancestors of the current element is also an ancestor of the root element (see [Bug#16321](#)).
- The following XPath functions are not supported:
 - `id()`
 - `lang()`
 - Prior to MySQL 5.1.8, the `last()` function was not supported (see [Bug#16318](#)).
 - `local-name()`
 - `name()`
 - `namespace-uri()`
 - `normalize-space()`
 - `starts-with()`
 - `string()`
 - `substring-after()`
 - `substring-before()`
 - `translate()`
- The following axes are not supported:
 - `following-sibling`
 - `following`
 - `preceding-sibling`
 - `preceding`

Beginning with MySQL 5.1.10, XPath expressions passed as arguments to `ExtractValue()` and `UpdateXML()` may contain the colon character (“:”) in element selectors, which enables their use with markup employing XML namespaces notation. For example:

```
mysql> SET @xml = '<a>111<b:c>222<d>333</d><e:f>444</e:f></b:c></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//e:f');
+-----+
| ExtractValue(@xml, '//e:f') |
+-----+
| 444                          |
+-----+
1 row in set (0.00 sec)

mysql> SELECT UpdateXML(@xml, '//b:c', '<g:h>555</g:h>');
+-----+
| UpdateXML(@xml, '//b:c', '<g:h>555</g:h>') |
+-----+
| <a>111<g:h>555</g:h></a>                  |
+-----+
1 row in set (0.00 sec)
```

This is similar in some respects to what is allowed by [Apache Xalan](#) and some other parsers, and is much simpler than requiring namespace declarations or the use of the `namespace-uri()` and `local-name()` functions.

Error handling. For both `ExtractValue()` and `UpdateXML()`, the XPath locator used must be valid and the XML to be searched must be well-formed. If the locator is invalid, an error is generated:

```
mysql> SELECT ExtractValue('<a>c</a><b/>', '&a');
ERROR 1105 (HY000): XPATH SYNTAX ERROR: '&a'
```

If `xml_frag` is invalid XML, then, beginning with MySQL 5.1.12, `NULL` is returned, and a warning is generated, as shown in this example:

```
mysql> SELECT ExtractValue('<a>c</a><b', '//@a');
+-----+
| ExtractValue('<a>c</a><b', '//@a') |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1523 | Incorrect XML value: 'parse error at line 1 pos 11: END-OF-INPUT unexpected ('>' wanted)' |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//@a');
+-----+
| ExtractValue('<a>c</a><b/>', '//@a') |
+-----+
| c |
+-----+
1 row in set (0.00 sec)
```

Prior to MySQL 5.1.12, an empty string was returned in such cases. ([Bug#18201](#))

Important

The replacement XML used as the third argument to `UpdateXML()` is *not* checked for well-formedness.

11.11. Other Functions

Name	Description
<code>AES_DECRYPT()</code>	Decrypt using AES
<code>AES_ENCRYPT()</code>	Encrypt using AES
<code>BENCHMARK()</code>	Repeatedly execute an expression
<code>BIT_COUNT()</code>	Return the number of bits that are set
<code>&</code>	Bitwise AND
<code> </code>	Bitwise OR
<code>^</code>	Bitwise XOR
<code>CHARSET()</code> (v4.1.0)	Return the character set of the argument
<code>COERCIBILITY()</code> (v4.1.1)	Return the collation coercibility value of the string argument
<code>COLLATION()</code> (v4.1.0)	Return the collation of the string argument
<code>COMPRESS()</code> (v4.1.1)	Return result as a binary string
<code>CONNECTION_ID()</code>	Return the connection ID (thread ID) for the connection
<code>CURRENT_USER()</code> , <code>CURRENT_USER</code>	Return the username and hostname combination
<code>DATABASE()</code>	Return the default (current) database name
<code>DECODE()</code>	Decodes a string encrypted using <code>ENCODE()</code>
<code>DEFAULT()</code>	Return the default value for a table column
<code>DES_DECRYPT()</code>	Decrypt a string
<code>DES_ENCRYPT()</code>	Decrypt a string
<code>ENCODE()</code>	Encode a string
<code>ENCRYPT()</code>	Encrypt a string
<code>FOUND_ROWS()</code>	For a <code>SELECT</code> with a <code>LIMIT</code> clause, the number of rows that would be returned were there no <code>LIMIT</code> clause
<code>GET_LOCK()</code>	Get a named lock
<code>INET_ATON()</code>	Return the numeric value of an IP address

Name	Description
INET_NTOA()	Return the IP address from a numeric value
IS_FREE_LOCK()	Checks whether the named lock is free
IS_USED_LOCK() (v4.1.0)	Checks whether the named lock is in use. Return connection identifier if true.
LAST_INSERT_ID()	Value of the AUTOINCREMENT column for the last INSERT
<<	Left shift
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
MD5()	Calculate MD5 checksum
NAME_CONST() (v5.0.12)	Causes the column to have the given name
OLD_PASSWORD() (v4.1)	Return the value of the old (pre-4.1) implementation of PASSWORD
PASSWORD()	Calculate and return a password string
RELEASE_LOCK()	Releases the named lock
>>	Right shift
ROW_COUNT() (v5.0.1)	The number of rows updated
SCHEMA() (v5.0.2)	A synonym for DATABASE()
SESSION_USER()	Synonym for USER()
SHA1() , SHA()	Calculate an SHA-1 160-bit checksum
SLEEP() (v5.0.12)	Sleep for a number of seconds
SYSTEM_USER()	Synonym for USER()
~	Invert bits
UNCOMPRESS() (v4.1.1)	Uncompress a string compressed
UNCOMPRESSED_LENGTH() (v4.1.1)	Return the length of a string before compression
USER()	Return the current username and hostname
UUID() (v4.1.2)	Return a Universal Unique Identifier (UUID)
VALUES() (v4.1.1)	Defines the values to be used during an INSERT
VERSION()	Returns a string that indicates the MySQL server version

11.11.1. Bit Functions

Name	Description
BIT_COUNT()	Return the number of bits that are set
&	Bitwise AND
 	Bitwise OR
^	Bitwise XOR
<<	Left shift
>>	Right shift
~	Invert bits

MySQL uses [BIGINT](#) (64-bit) arithmetic for bit operations, so these operators have a maximum range of 64 bits.

- [|](#)

Bitwise OR:

```
mysql> SELECT 29 | 15;
-> 31
```

The result is an unsigned 64-bit integer.

- **&**

Bitwise AND:

```
mysql> SELECT 29 & 15;
-> 13
```

The result is an unsigned 64-bit integer.

- **^**

Bitwise XOR:

```
mysql> SELECT 1 ^ 1;
-> 0
mysql> SELECT 1 ^ 0;
-> 1
mysql> SELECT 11 ^ 3;
-> 8
```

The result is an unsigned 64-bit integer.

- **<<**

Shifts a longlong (**BIGINT**) number to the left.

```
mysql> SELECT 1 << 2;
-> 4
```

The result is an unsigned 64-bit integer.

- **>>**

Shifts a longlong (**BIGINT**) number to the right.

```
mysql> SELECT 4 >> 2;
-> 1
```

The result is an unsigned 64-bit integer.

- **~**

Invert all bits.

```
mysql> SELECT 5 & ~1;
-> 4
```

The result is an unsigned 64-bit integer.

- **BIT_COUNT(*N*)**

Returns the number of bits that are set in the argument *N*.

```
mysql> SELECT BIT_COUNT(29), BIT_COUNT(b'101010');
-> 4, 3
```

11.11.2. Encryption and Compression Functions

Name	Description
AES_DECRYPT()	Decrypt using AES
AES_ENCRYPT()	Encrypt using AES

Name	Description
COMPRESS() (v4.1.1)	Return result as a binary string
DECODE()	Decodes a string encrypted using ENCODE()
DES_DECRYPT()	Decrypt a string
DES_ENCRYPT()	Encrypt a string
ENCODE()	Encode a string
ENCRYPT()	Encrypt a string
MD5()	Calculate MD5 checksum
OLD_PASSWORD() (v4.1)	Return the value of the old (pre-4.1) implementation of PASSWORD
PASSWORD()	Calculate and return a password string
SHA1() , SHA()	Calculate an SHA-1 160-bit checksum
UNCOMPRESS() (v4.1.1)	Uncompress a string compressed
UNCOMPRESSED_LENGTH() (v4.1.1)	Return the length of a string before compression

The functions in this section perform encryption and decryption, and compression and uncompression:

Compression or encryption	Uncompression or decryption
AES_ENCRYPT()	AES_DECRYPT()
COMPRESS()	UNCOMPRESS()
ENCODE()	DECODE()
DES_ENCRYPT()	DES_DECRYPT()
ENCRYPT()	Not available
MD5()	Not available
OLD_PASSWORD()	Not available
PASSWORD()	Not available
SHA() or SHA1()	Not available
Not available	UNCOMPRESSED_LENGTH()

Note

The encryption and compression functions return binary strings. For many of these functions, the result might contain arbitrary byte values. If you want to store these results, use a [BLOB](#) column rather than a [CHAR](#) or [VARCHAR](#) column to avoid potential problems with trailing space removal that would change data values.

Note

Exploits for the MD5 and SHA-1 algorithms have become known. You may wish to consider using one of the other encryption functions described in this section instead.

- [AES_ENCRYPT\(str, key_str\)](#), [AES_DECRYPT\(encrypt_str, key_str\)](#)

These functions allow encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as “Rijndael.” Encoding with a 128-bit key length is used, but you can extend it up to 256 bits by modifying the source. We chose 128 bits because it is much faster and it is secure enough for most purposes.

[AES_ENCRYPT\(\)](#) encrypts a string and returns a binary string. [AES_DECRYPT\(\)](#) decrypts the encrypted string and returns the original string. The input arguments may be any length. If either argument is [NULL](#), the result of this function is also [NULL](#).

Because AES is a block-level algorithm, padding is used to encode uneven length strings and so the result string length may be calculated using this formula:

$$16 \times (\text{trunc}(\text{string\_length} / 16) + 1)$$

If `AES_DECRYPT()` detects invalid data or incorrect padding, it returns `NULL`. However, it is possible for `AES_DECRYPT()` to return a non-`NULL` value (possibly garbage) if the input data or the key is invalid.

You can use the AES functions to store data in an encrypted form by modifying your queries:

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));
```

`AES_ENCRYPT()` and `AES_DECRYPT()` can be considered the most cryptographically secure encryption functions currently available in MySQL.

- `COMPRESS(string_to_compress)`

Compresses a string and returns the result as a binary string. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`. The compressed string can be uncompressed with `UNCOMPRESS()`.

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
-> 21
mysql> SELECT LENGTH(COMPRESS(''));
-> 0
mysql> SELECT LENGTH(COMPRESS('a'));
-> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
-> 15
```

The compressed string contents are stored the following way:

- Empty strings are stored as empty strings.
- Non-empty strings are stored as a four-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra `'.'` character is added to avoid problems with endspace trimming should the result be stored in a `CHAR` or `VARCHAR` column. (Use of `CHAR` or `VARCHAR` to store compressed strings is not recommended. It is better to use a `BLOB` column instead.)
- `DECODE(encrypt_str,pass_str)`

Decrypts the encrypted string `encrypt_str` using `pass_str` as the password. `encrypt_str` should be a string returned from `ENCODE()`.

- `ENCODE(str,pass_str)`

Encrypt `str` using `pass_str` as the password. To decrypt the result, use `DECODE()`.

The result is a binary string of the same length as `str`.

The strength of the encryption is based on how good the random generator is. It should suffice for short strings.

- `DES_DECRYPT(encrypt_str[,key_str])`

Decrypts a string encrypted with `DES_ENCRYPT()`. If an error occurs, this function returns `NULL`.

Note that this function works only if MySQL has been configured with SSL support. See [Section 5.8.7, “Using Secure Connections”](#).

If no `key_str` argument is given, `DES_DECRYPT()` examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the `SUPER` privilege. The key file can be specified with the `--des-key-file` server option.

If you pass this function a `key_str` argument, that string is used as the key for decrypting the message.

If the `encrypt_str` argument does not appear to be an encrypted string, MySQL returns the given `encrypt_str`.

- `DES_ENCRYPT(str[,key_num|key_str])`

Encrypts the string with the given key using the Triple-DES algorithm.

Note that this function works only if MySQL has been configured with SSL support. See [Section 5.8.7, “Using Secure Connections”](#).

The encryption key to use is chosen based on the second argument to `DES_ENCRYPT()`, if one was given:

Argument	Description
No argument	The first key from the DES key file is used.
<i>key_num</i>	The given key number (0-9) from the DES key file is used.
<i>key_str</i>	The given key string is used to encrypt <i>str</i> .

The key file can be specified with the `--des-key-file` server option.

The return string is a binary string where the first character is `CHAR(128 | key_num)`. If an error occurs, `DES_ENCRYPT()` returns `NULL`.

The 128 is added to make it easier to recognize an encrypted key. If you use a string key, *key_num* is 127.

The string length for the result is given by this formula:

```
new_len = orig_len + (8 - (orig_len % 8)) + 1
```

Each line in the DES key file has the following format:

```
key_num des_key_str
```

Each *key_num* value must be a number in the range from 0 to 9. Lines in the file may be in any order. *des_key_str* is the string that is used to encrypt the message. There should be at least one space between the number and the key. The first key is the default key that is used if you do not specify any key argument to `DES_ENCRYPT()`.

You can tell MySQL to read new key values from the key file with the `FLUSH DES_KEY_FILE` statement. This requires the `RELOAD` privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

```
mysql> SELECT customer_address FROM customer_table
> WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

- `ENCRYPT(str [, salt])`

Encrypts *str* using the Unix `crypt()` system call and returns a binary string. The *salt* argument should be a string with at least two characters. If no *salt* argument is given, a random value is used.

```
mysql> SELECT ENCRYPT('hello');
-> 'VxuFAJXVAR0c'
```

`ENCRYPT()` ignores all but the first eight characters of *str*, at least on some systems. This behavior is determined by the implementation of the underlying `crypt()` system call.

The use of `ENCRYPT()` with multi-byte character sets other than `utf8` is not recommended because the system call expects a string terminated by a zero byte.

If `crypt()` is not available on your system (as is the case with Windows), `ENCRYPT()` always returns `NULL`.

- `MD5(str)`

Calculates an MD5 128-bit checksum for the string. The value is returned as a binary string of 32 hex digits, or `NULL` if the argument was `NULL`. The return value can, for example, be used as a hash key.

```
mysql> SELECT MD5('testing');
-> 'ae2b1fca515949e5d54fb22b8ed95575'
```

This is the “RSA Data Security, Inc. MD5 Message-Digest Algorithm.”

If you want to convert the value to uppercase, see the description of binary string conversion given in the entry for the [BINARY](#) operator in [Section 11.9, “Cast Functions and Operators”](#).

See the note regarding the MD5 algorithm at the beginning this section.

- [OLD_PASSWORD\(*str*\)](#)

[OLD_PASSWORD\(\)](#) was added to MySQL when the implementation of [PASSWORD\(\)](#) was changed to improve security. [OLD_PASSWORD\(\)](#) returns the value of the old (pre-4.1) implementation of [PASSWORD\(\)](#) as a binary string, and is intended to permit you to reset passwords for any pre-4.1 clients that need to connect to your version 5.1 MySQL server without locking them out. See [Section 5.7.9, “Password Hashing as of MySQL 4.1”](#).

- [PASSWORD\(*str*\)](#)

Calculates and returns a password string from the plaintext password *str* and returns a binary string, or [NULL](#) if the argument was [NULL](#). This is the function that is used for encrypting MySQL passwords for storage in the [Password](#) column of the [user](#) grant table.

```
mysql> SELECT PASSWORD('badpwd');
-> '*AAB3E285149C0135D51A520E1940DD3263DC008C'
```

[PASSWORD\(\)](#) encryption is one-way (not reversible).

[PASSWORD\(\)](#) does not perform password encryption in the same way that Unix passwords are encrypted. See [ENCRYPT\(\)](#).

Note

The [PASSWORD\(\)](#) function is used by the authentication system in MySQL Server; you should *not* use it in your own applications. For that purpose, consider [MD5\(\)](#) or [SHA1\(\)](#) instead. Also see [RFC 2195, section 2 \(Challenge-Response Authentication Mechanism \(CRAM\)\)](#), for more information about handling passwords and authentication securely in your applications.

- [SHA1\(*str*\)](#), [SHA\(*str*\)](#)

Calculates an SHA-1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a binary string of 40 hex digits, or [NULL](#) if the argument was [NULL](#). One of the possible uses for this function is as a hash key. You can also use it as a cryptographic function for storing passwords. [SHA\(\)](#) is synonymous with [SHA1\(\)](#).

```
mysql> SELECT SHA1('abc');
-> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

[SHA1\(\)](#) can be considered a cryptographically more secure equivalent of [MD5\(\)](#). However, see the note regarding the MD5 and SHA-1 algorithms at the beginning this section.

- [UNCOMPRESS\(*string_to_uncompress*\)](#)

Uncompresses a string compressed by the [COMPRESS\(\)](#) function. If the argument is not a compressed value, the result is [NULL](#). This function requires MySQL to have been compiled with a compression library such as [zlib](#). Otherwise, the return value is always [NULL](#).

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
-> 'any string'
mysql> SELECT UNCOMPRESS('any string');
-> NULL
```

- [UNCOMPRESSED_LENGTH\(*compressed_string*\)](#)

Returns the length that the compressed string had before being compressed.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a', 30)));
-> 30
```

11.11.3. Information Functions

Name	Description
<code>BENCHMARK()</code>	Repeatedly execute an expression
<code>CHARSET()</code> (v4.1.0)	Return the character set of the argument
<code>COERCIBILITY()</code> (v4.1.1)	Return the collation coercibility value of the string argument
<code>COLLATION()</code> (v4.1.0)	Return the collation of the string argument
<code>CONNECTION_ID()</code>	Return the connection ID (thread ID) for the connection
<code>CURRENT_USER()</code> , <code>CURRENT_USER</code>	Return the username and hostname combination
<code>DATABASE()</code>	Return the default (current) database name
<code>FOUND_ROWS()</code>	For a <code>SELECT</code> with a <code>LIMIT</code> clause, the number of rows that would be returned were there no <code>LIMIT</code> clause
<code>LAST_INSERT_ID()</code>	Value of the <code>AUTOINCREMENT</code> column for the last <code>INSERT</code>
<code>ROW_COUNT()</code> (v5.0.1)	The number of rows updated
<code>SCHEMA()</code> (v5.0.2)	A synonym for <code>DATABASE()</code>
<code>SESSION_USER()</code>	Synonym for <code>USER()</code>
<code>SYSTEM_USER()</code>	Synonym for <code>USER()</code>
<code>USER()</code>	Return the current username and hostname
<code>VERSION()</code>	Returns a string that indicates the MySQL server version

- `BENCHMARK(count, expr)`

The `BENCHMARK()` function executes the expression *expr* repeatedly *count* times. It may be used to time how quickly MySQL processes the expression. The result value is always 0. The intended use is from within the `mysql` client, which reports query execution times:

```
mysql> SELECT BENCHMARK(1000000, ENCODE('hello', 'goodbye'));
+-----+
| BENCHMARK(1000000, ENCODE('hello', 'goodbye')) |
+-----+
| 0 |
+-----+
1 row in set (4.74 sec)
```

The time reported is elapsed time on the client end, not CPU time on the server end. It is advisable to execute `BENCHMARK()` several times, and to interpret the result with regard to how heavily loaded the server machine is.

`BENCHMARK()` is intended for measuring the runtime performance of scalar expressions, which has some significant implications for the way that you use it and interpret the results:

- Only scalar expressions can be used. Although the expression can be a subquery, it must return a single column and at most a single row. For example, `BENCHMARK(10, (SELECT * FROM t))` will fail if the table `t` has more than one column or more than one row.
- Executing a `SELECT expr` statement *N* times differs from executing `SELECT BENCHMARK(N, expr)` in terms of the amount of overhead involved. The two have very different execution profiles and you should not expect them to take the same amount of time. The former involves the parser, optimizer, table locking, and runtime evaluation *N* times each. The latter involves only runtime evaluation *N* times, and all the other components just once. Memory structures already allocated are reused, and runtime optimizations such as local caching of results already evaluated for aggregate functions can alter the results. Use of `BENCHMARK()` thus measures performance of the runtime component by giving more weight to that component and removing the “noise” introduced by the network, parser, optimizer, and so forth.
- `CHARSET(str)`

Returns the character set of the string argument.

```
mysql> SELECT CHARSET('abc');
-> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
```

```
mysql> -> 'utf8'
mysql> SELECT CHARSET(USER());
mysql> -> 'utf8'
```

- **COERCIBILITY(*str*)**

Returns the collation coercibility value of the string argument.

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
mysql> -> 0
mysql> SELECT COERCIBILITY(USER());
mysql> -> 3
mysql> SELECT COERCIBILITY('abc');
```

The return values have the meanings shown in the following table. Lower values have higher precedence.

Coercibility	Meaning	Example
0	Explicit collation	Value with COLLATE clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value, stored routine parameter or local variable
3	System constant	USER() return value
4	Coercible	Literal string
5	Ignorable	NULL or an expression derived from NULL

- **COLLATION(*str*)**

Returns the collation of the string argument.

```
mysql> SELECT COLLATION('abc');
mysql> -> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
mysql> -> 'utf8_general_ci'
```

- **CONNECTION_ID()**

Returns the connection ID (thread ID) for the connection. Every connection has an ID that is unique among the set of currently connected clients.

```
mysql> SELECT CONNECTION_ID();
mysql> -> 23786
```

- **CURRENT_USER, CURRENT_USER()**

Returns the username and hostname combination for the MySQL account that the server used to authenticate the current client. This account determines your access privileges. Within a stored routine that is defined with the **SQL SECURITY DEFINER** characteristic, **CURRENT_USER()** returns the creator of the routine. The return value is a string in the **utf8** character set.

The value of **CURRENT_USER()** can differ from the value of **USER()**.

```
mysql> SELECT USER();
mysql> -> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user ''@'localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
mysql> -> '@localhost'
```

The example illustrates that although the client specified a username of **davida** (as indicated by the value of the **USER()** function), the server authenticated the client using an anonymous user account (as seen by the empty username part of the **CURRENT_USER()** value). One way this might occur is that there is no account listed in the grant tables for **davida**.

- **DATABASE()**

Returns the default (current) database name as a string in the `utf8` character set. If there is no default database, `DATABASE()` returns `NULL`. Within a stored routine, the default database is the database that the routine is associated with, which is not necessarily the same as the database that is the default in the calling context.

```
mysql> SELECT DATABASE();
-> 'test'
```

If there is no default database, `DATABASE()` returns `NULL`.

- `FOUND_ROWS()`

A `SELECT` statement may include a `LIMIT` clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the `LIMIT`, but without running the statement again. To obtain this row count, include a `SQL_CALC_FOUND_ROWS` option in the `SELECT` statement, and then invoke `FOUND_ROWS()` afterward:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
-> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

The second `SELECT` returns a number indicating how many rows the first `SELECT` would have returned had it been written without the `LIMIT` clause.

In the absence of the `SQL_CALC_FOUND_ROWS` option in the most recent `SELECT` statement, `FOUND_ROWS()` returns the number of rows in the result set returned by that statement.

The row count available through `FOUND_ROWS()` is transient and not intended to be available past the statement following the `SELECT SQL_CALC_FOUND_ROWS` statement. If you need to refer to the value later, save it:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;
mysql> SET @rows = FOUND_ROWS();
```

If you are using `SELECT SQL_CALC_FOUND_ROWS`, MySQL must calculate how many rows are in the full result set. However, this is faster than running the query again without `LIMIT`, because the result set need not be sent to the client.

`SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` can be useful in situations when you want to restrict the number of rows that a query returns, but also determine the number of rows in the full result set without running the query again. An example is a Web script that presents a paged display containing links to the pages that show other sections of a search result. Using `FOUND_ROWS()` allows you to determine how many other pages are needed for the rest of the result.

The use of `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` is more complex for `UNION` statements than for simple `SELECT` statements, because `LIMIT` may occur at multiple places in a `UNION`. It may be applied to individual `SELECT` statements in the `UNION`, or global to the `UNION` result as a whole.

The intent of `SQL_CALC_FOUND_ROWS` for `UNION` is that it should return the row count that would be returned without a global `LIMIT`. The conditions for use of `SQL_CALC_FOUND_ROWS` with `UNION` are:

- The `SQL_CALC_FOUND_ROWS` keyword must appear in the first `SELECT` of the `UNION`.
- The value of `FOUND_ROWS()` is exact only if `UNION ALL` is used. If `UNION` without `ALL` is used, duplicate removal occurs and the value of `FOUND_ROWS()` is only approximate.
- If no `LIMIT` is present in the `UNION`, `SQL_CALC_FOUND_ROWS` is ignored and returns the number of rows in the temporary table that is created to process the `UNION`.
- `LAST_INSERT_ID()`, `LAST_INSERT_ID(expr)`

For MySQL 5.1.12 and later, `LAST_INSERT_ID()` (no arguments) returns the *first* automatically generated value *successfully* inserted for an `AUTO_INCREMENT` column as a result of the most recently executed `INSERT` statement. The value of `LAST_INSERT_ID()` remains unchanged if no rows are successfully inserted.

For example, after inserting a row that generates an `AUTO_INCREMENT` value, you can get the value like this:

```
mysql> SELECT LAST_INSERT_ID();
-> 195
```

In MySQL 5.1.11 and earlier, `LAST_INSERT_ID()` (no arguments) returns the *first* automatically generated value if any rows were successfully inserted or updated. This means that the returned value could be a value that was not successfully inserted into the table. If no rows were successfully inserted, `LAST_INSERT_ID()` returns 0.

The value of `LAST_INSERT_ID()` will be consistent across all versions if all rows in the `INSERT` or `UPDATE` statement were successful.

The currently executing statement does not affect the value of `LAST_INSERT_ID()`. Suppose that you generate an `AUTO_INCREMENT` value with one statement, and then refer to `LAST_INSERT_ID()` in a multiple-row `INSERT` statement that inserts rows into a table with its own `AUTO_INCREMENT` column. The value of `LAST_INSERT_ID()` will remain stable in the second statement; its value for the second and later rows is not affected by the earlier row insertions. (However, if you mix references to `LAST_INSERT_ID()` and `LAST_INSERT_ID(expr)`, the effect is undefined.)

If the previous statement returned an error, the value of `LAST_INSERT_ID()` is undefined. For transactional tables, if the statement is rolled back due to an error, the value of `LAST_INSERT_ID()` is left undefined. For manual `ROLLBACK`, the value of `LAST_INSERT_ID()` is not restored to that before the transaction; it remains as it was at the point of the `ROLLBACK`.

Within the body of a stored routine (procedure or function) or a trigger, the value of `LAST_INSERT_ID()` changes the same way as for statements executed outside the body of these kinds of objects. The effect of a stored routine or trigger upon the value of `LAST_INSERT_ID()` that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of `LAST_INSERT_ID()`, the changed value will be seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements will not see a changed value.

The ID that was generated is maintained in the server on a *per-connection basis*. This means that the value returned by the function to a given client is the first `AUTO_INCREMENT` value generated for most recent statement affecting an `AUTO_INCREMENT` column *by that client*. This value cannot be affected by other clients, even if they generate `AUTO_INCREMENT` values of their own. This behavior ensures that each client can retrieve its own ID without concern for the activity of other clients, and without the need for locks or transactions.

The value of `LAST_INSERT_ID()` is not changed if you set the `AUTO_INCREMENT` column of a row to a non-“magic” value (that is, a value that is not `NULL` and not 0).

Important

If you insert multiple rows using a single `INSERT` statement, `LAST_INSERT_ID()` returns the value generated for the *first* inserted row *only*. The reason for this is to make it possible to reproduce easily the same `INSERT` statement against some other server.

For example:

```
mysql> USE test;
Database changed
mysql> CREATE TABLE t (
->   id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
->   name VARCHAR(10) NOT NULL
-> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t VALUES (NULL, 'Bob');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t;
+----+-----+
| id | name |
+----+-----+
|  1 | Bob  |
+----+-----+
1 row in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|                1 |
+-----+
1 row in set (0.00 sec)

mysql> INSERT INTO t VALUES
```

```
-> (NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t;
+----+-----+
| id | name |
+----+-----+
| 1  | Bob  |
| 2  | Mary |
| 3  | Jane |
| 4  | Lisa |
+----+-----+
4 rows in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
| 2                |
+-----+
1 row in set (0.00 sec)
```

Although the second `INSERT` statement inserted three new rows into `t`, the ID generated for the first of these rows was `2`, and it is this value that is returned by `LAST_INSERT_ID()` for the following `SELECT` statement.

If you use `INSERT IGNORE` and the row is ignored, the `AUTO_INCREMENT` counter is not incremented and `LAST_INSERT_ID()` returns `0`, which reflects that no row was inserted.

If `expr` is given as an argument to `LAST_INSERT_ID()`, the value of the argument is returned by the function and is remembered as the next value to be returned by `LAST_INSERT_ID()`. This can be used to simulate sequences:

1. Create a table to hold the sequence counter and initialize it:

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
```

2. Use the table to generate sequence numbers like this:

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
mysql> SELECT LAST_INSERT_ID();
```

The `UPDATE` statement increments the sequence counter and causes the next call to `LAST_INSERT_ID()` to return the updated value. The `SELECT` statement retrieves that value. The `mysql_insert_id()` C API function can also be used to get the value. See [Section 25.2.3.37](#), “`mysql_insert_id()`”.

You can generate sequences without calling `LAST_INSERT_ID()`, but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. It is multi-user safe because multiple clients can issue the `UPDATE` statement and get their own sequence value with the `SELECT` statement (or `mysql_insert_id()`), without affecting or being affected by other clients that generate their own sequence values.

Note that `mysql_insert_id()` is only updated after `INSERT` and `UPDATE` statements, so you cannot use the C API function to retrieve the value for `LAST_INSERT_ID(expr)` after executing other SQL statements like `SELECT` or `SET`.

- **`ROW_COUNT()`**

`ROW_COUNT()` returns the number of rows updated, inserted, or deleted by the preceding statement. This is the same as the row count that the `mysql` client displays and the value from the `mysql_affected_rows()` C API function.

```
mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
| 3          |
+-----+
1 row in set (0.00 sec)

mysql> DELETE FROM t WHERE i IN(1,2);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT ROW_COUNT();
```

```
+-----+
| ROW_COUNT() |
+-----+
|          2 |
+-----+
1 row in set (0.00 sec)
```

- **SCHEMA()**

This function is a synonym for **DATABASE()**.

- **SESSION_USER()**

SESSION_USER() is a synonym for **USER()**.

- **SYSTEM_USER()**

SYSTEM_USER() is a synonym for **USER()**.

- **USER()**

Returns the current MySQL username and hostname as a string in the **utf8** character set.

```
mysql> SELECT USER();
-> 'davida@localhost'
```

The value indicates the username you specified when connecting to the server, and the client host from which you connected. The value can be different from that of **CURRENT_USER()**.

You can extract only the username part like this:

```
mysql> SELECT SUBSTRING_INDEX(USER(), '@', 1);
-> 'davida'
```

- **VERSION()**

Returns a string that indicates the MySQL server version. The string uses the **utf8** character set.

```
mysql> SELECT VERSION();
-> '5.1.21-beta-standard'
```

Note that if your version string ends with **-log** this means that logging is enabled.

11.11.4. Miscellaneous Functions

Name	Description
DEFAULT()	Return the default value for a table column
GET_LOCK()	Get a named lock
INET_ATON()	Return the numeric value of an IP address
INET_NTOA()	Return the IP address from a numeric value
IS_FREE_LOCK()	Checks whether the named lock is free
IS_USED_LOCK() (v4.1.0)	Checks whether the named lock is in use. Return connection identifier if true.
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
NAME_CONST() (v5.0.12)	Causes the column to have the given name
RELEASE_LOCK()	Releases the named lock
SLEEP() (v5.0.12)	Sleep for a number of seconds
UUID() (v4.1.2)	Return a Universal Unique Identifier (UUID)
VALUES() (v4.1.1)	Defines the values to be used during an INSERT

- `DEFAULT(col_name)`

Returns the default value for a table column. An error results if the column has no default value.

```
mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
```

- `FORMAT(X,D)`

Formats the number *X* to a format like '*#*,*###*,*###.###*', rounded to *D* decimal places, and returns the result as a string. For details, see [Section 11.4, “String Functions”](#).

- `GET_LOCK(str,timeout)`

Tries to obtain a lock with a name given by the string *str*, using a timeout of *timeout* seconds. Returns **1** if the lock was obtained successfully, **0** if the attempt timed out (for example, because another client has previously locked the name), or **NULL** if an error occurred (such as running out of memory or the thread was killed with `mysqladmin kill`). If you have a lock obtained with `GET_LOCK()`, it is released when you execute `RELEASE_LOCK()`, execute a new `GET_LOCK()`, or your connection terminates (either normally or abnormally). Locks obtained with `GET_LOCK()` do not interact with transactions. That is, committing a transaction does not release any such locks obtained during the transaction.

This function can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked by one client, `GET_LOCK()` blocks any request by another client for a lock with the same name. This allows clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also allows a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form *db_name.str* or *app_name.str*.

```
mysql> SELECT GET_LOCK('lock1',10);
-> 1
mysql> SELECT IS_FREE_LOCK('lock2');
-> 1
mysql> SELECT GET_LOCK('lock2',10);
-> 1
mysql> SELECT RELEASE_LOCK('lock2');
-> 1
mysql> SELECT RELEASE_LOCK('lock1');
-> NULL
```

The second `RELEASE_LOCK()` call returns **NULL** because the lock '*lock1*' was automatically released by the second `GET_LOCK()` call.

Note: If a client attempts to acquire a lock that is already held by another client, it blocks according to the *timeout* argument. If the blocked client terminates, its thread does not die until the lock request times out. This is a known bug.

- `INET_ATON(expr)`

Given the dotted-quad representation of a network address as a string, returns an integer that represents the numeric value of the address. Addresses may be 4- or 8-byte addresses.

```
mysql> SELECT INET_ATON('209.207.224.40');
-> 3520061480
```

The generated number is always in network byte order. For the example just shown, the number is calculated as $209 \times 256^3 + 207 \times 256^2 + 224 \times 256 + 40$.

`INET_ATON()` also understands short-form IP addresses:

```
mysql> SELECT INET_ATON('127.0.0.1'), INET_ATON('127.1');
-> 2130706433, 2130706433
```

Note

When storing values generated by `INET_ATON()`, it is recommended that you use an `INT UNSIGNED` column. If you use a (signed) `INT` column, values corresponding to IP addresses for which the first octet is greater than 127 cannot be stored correctly. See [Section 10.2, “Numeric Types”](#).

- `INET_NTOA(expr)`

Given a numeric network address (4 or 8 byte), returns the dotted-quad representation of the address as a string.

```
mysql> SELECT INET_NTOA(3520061480);
-> '209.207.224.40'
```

- **IS_FREE_LOCK(*str*)**

Checks whether the lock named *str* is free to use (that is, not locked). Returns **1** if the lock is free (no one is using the lock), **0** if the lock is in use, and **NULL** if an error occurs (such as an incorrect argument).

- **IS_USED_LOCK(*str*)**

Checks whether the lock named *str* is in use (that is, locked). If so, it returns the connection identifier of the client that holds the lock. Otherwise, it returns **NULL**.

- **MASTER_POS_WAIT(*log_name*, *log_pos*[, *timeout*])**

This function is useful for control of master/slave synchronization. It blocks until the slave has read and applied all updates up to the specified position in the master log. The return value is the number of log events the slave had to wait for to advance to the specified position. The function returns **NULL** if the slave SQL thread is not started, the slave's master information is not initialized, the arguments are incorrect, or an error occurs. It returns **-1** if the timeout has been exceeded. If the slave SQL thread stops while **MASTER_POS_WAIT()** is waiting, the function returns **NULL**. If the slave is past the specified position, the function returns immediately.

If a *timeout* value is specified, **MASTER_POS_WAIT()** stops waiting when *timeout* seconds have elapsed. *timeout* must be greater than 0; a zero or negative *timeout* means no timeout.

- **NAME_CONST(*name*, *value*)**

Returns the given value. When used to produce a result set column, **NAME_CONST()** causes the column to have the given name.

```
mysql> SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
|      14 |
+-----+
```

This function was added in MySQL 5.0.12. It is for internal use only. The server uses it when writing statements from stored routines that contain references to local routine variables, as described in [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#). You might see this function in the output from **mysqlbinlog**.

- **RELEASE_LOCK(*str*)**

Releases the lock named by the string *str* that was obtained with **GET_LOCK()**. Returns **1** if the lock was released, **0** if the lock was not established by this thread (in which case the lock is not released), and **NULL** if the named lock did not exist. The lock does not exist if it was never obtained by a call to **GET_LOCK()** or if it has previously been released.

The **DO** statement is convenient to use with **RELEASE_LOCK()**. See [Section 12.2.2, “DO Syntax”](#).

- **SLEEP(*duration*)**

Sleeps (pauses) for the number of seconds given by the *duration* argument, then returns 0. If **SLEEP()** is interrupted, it returns 1. The duration may have a fractional part given in microseconds.

- **UUID()**

Returns a Universal Unique Identifier (UUID) generated according to “DCE 1.1: Remote Procedure Call” (Appendix A) CAE (Common Applications Environment) Specifications published by The Open Group in October 1997 (Document Number C706, <http://www.opengroup.org/public/pubs/catalog/c706.htm>).

A UUID is designed as a number that is globally unique in space and time. Two calls to **UUID()** are expected to generate two different values, even if these calls are performed on two separate computers that are not connected to each other.

A UUID is a 128-bit number represented by a string of five hexadecimal numbers in **aaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee** format:

- The first three numbers are generated from a timestamp.
- The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).
- The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host computer has no Ethernet card, or we do not know how to find the hardware address of an interface on your operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have very low probability.

Currently, the MAC address of an interface is taken into account only on FreeBSD and Linux. On other operating systems, MySQL uses a randomly generated 48-bit number.

```
mysql> SELECT UUID();
-> '6ccd780c-baba-1026-9564-0040f4311e29'
```

Note that `UUID()` does not yet work with replication.

- `VALUES(col_name)`

In an `INSERT ... ON DUPLICATE KEY UPDATE` statement, you can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of `col_name` that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` function is meaningful only in `INSERT ... ON DUPLICATE KEY UPDATE` statements and returns `NULL` otherwise. [Section 12.2.4.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”](#).

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3), (4,5,6)
-> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

11.12. Functions and Modifiers for Use with **GROUP BY** Clauses

11.12.1. **GROUP BY** (Aggregate) Functions

Name	Description
<code>AVG()</code>	Return the average value of the argument
<code>BIT_AND()</code>	Return bitwise and
<code>BIT_OR()</code>	Return bitwise or
<code>BIT_XOR()</code> (v4.1.1)	Return bitwise xor
<code>COUNT(DISTINCT)</code>	Return the count of a number of different values
<code>COUNT()</code>	Return a count of the number of rows returned
<code>GROUP_CONCAT()</code> (v4.1)	Return a concatenated string
<code>MAX()</code>	Return the maximum value
<code>MIN()</code>	Return the minimum value
<code>STD(), STDDEV()</code>	Return the population standard deviation
<code>STDDEV_POP()</code> (v5.0.3)	Return the population standard deviation
<code>STDDEV_SAMP()</code> (v5.0.3)	Return the sample standard deviation
<code>SUM()</code>	Return the sum
<code>VAR_POP()</code> (v5.0.3)	Return the population standard variance
<code>VAR_SAMP()</code> (v5.0.3)	Return the sample variance
<code>VARIANCE()</code> (v4.1)	Return the population standard variance

This section describes group (aggregate) functions that operate on sets of values. Unless otherwise stated, group functions ignore `NULL` values.

If you use a group function in a statement containing no **GROUP BY** clause, it is equivalent to grouping on all rows.

For numeric arguments, the variance and standard deviation functions return a **DOUBLE** value. The **SUM()** and **AVG()** functions return a **DECIMAL** value for exact-value arguments (integer or **DECIMAL**), and a **DOUBLE** value for approximate-value arguments (**FLOAT** or **DOUBLE**).

The **SUM()** and **AVG()** aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first non-numeric character.) To work around this problem, you can convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

Functions such as **SUM()** or **AVG()** that expect a numeric argument cast the argument to a number if necessary. For **SET** or **ENUM** values, the cast operation causes the underlying numeric value to be used.

- **AVG([DISTINCT] *expr*)**

Returns the average value of *expr*. The **DISTINCT** option can be used to return the average of the distinct values of *expr*.

AVG() returns **NULL** if there were no matching rows.

```
mysql> SELECT student_name, AVG(test_score)
->      FROM student
->      GROUP BY student_name;
```

- **BIT_AND(*expr*)**

Returns the bitwise **AND** of all bits in *expr*. The calculation is performed with 64-bit (**BIGINT**) precision.

This function returns **18446744073709551615** if there were no matching rows. (This is the value of an unsigned **BIGINT** value with all bits set to 1.)

- **BIT_OR(*expr*)**

Returns the bitwise **OR** of all bits in *expr*. The calculation is performed with 64-bit (**BIGINT**) precision.

This function returns **0** if there were no matching rows.

- **BIT_XOR(*expr*)**

Returns the bitwise **XOR** of all bits in *expr*. The calculation is performed with 64-bit (**BIGINT**) precision.

This function returns **0** if there were no matching rows.

- **COUNT(*expr*)**

Returns a count of the number of non-**NULL** values of *expr* in the rows retrieved by a **SELECT** statement. The result is a **BIGINT** value.

COUNT() returns **0** if there were no matching rows.

```
mysql> SELECT student.student_name, COUNT(*)
->      FROM student, course
->      WHERE student.student_id=course.student_id
->      GROUP BY student_name;
```

COUNT(*) is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain **NULL** values.

COUNT(*) is optimized to return very quickly if the **SELECT** retrieves from one table, no other columns are retrieved, and there is no **WHERE** clause. For example:

```
mysql> SELECT COUNT(*) FROM student;
```


This optimization applies only to **MyISAM** tables only, because an exact row count is stored for this storage engine and can be accessed very quickly. For transactional storage engines such as **InnoDB**, storing an exact row count is more problematic because multiple transactions may be occurring, each of which may affect the count.

- **COUNT(DISTINCT *expr*, [*expr*...])**

Returns a count of the number of different non-**NULL** values.

COUNT(DISTINCT) returns 0 if there were no matching rows.

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

In MySQL, you can obtain the number of distinct expression combinations that do not contain **NULL** by giving a list of expressions. In standard SQL, you would have to do a concatenation of all expressions inside **COUNT(DISTINCT ...)**.

- **GROUP_CONCAT(*expr*)**

This function returns a string result with the concatenated non-**NULL** values from a group. It returns **NULL** if there are no non-**NULL** values. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [, expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
              [ASC | DESC] [, col_name ...]]
             [SEPARATOR str_val])
```

```
mysql> SELECT student_name,
->        GROUP_CONCAT(test_score)
->        FROM student
->        GROUP BY student_name;
```

Or:

```
mysql> SELECT student_name,
->        GROUP_CONCAT(DISTINCT test_score
->                      ORDER BY test_score DESC SEPARATOR ' ')
->        FROM student
->        GROUP BY student_name;
```

In MySQL, you can get the concatenated values of expression combinations. You can eliminate duplicate values by using **DISTINCT**. If you want to sort values in the result, you should use **ORDER BY** clause. To sort in reverse order, add the **DESC** (descending) keyword to the name of the column you are sorting by in the **ORDER BY** clause. The default is ascending order; this may be specified explicitly using the **ASC** keyword. **SEPARATOR** is followed by the string value that should be inserted between values of result. The default is a comma (','). You can eliminate the separator altogether by specifying **SEPARATOR ''**.

The result is truncated to the maximum length that is given by the **group_concat_max_len** system variable, which has a default value of 1024. The value can be set higher, although the maximum effective length of the return value is constrained by the value of **max_allowed_packet**. The syntax to change the value of **group_concat_max_len** at runtime is as follows, where *val* is an unsigned integer:

```
SET [SESSION | GLOBAL] group_concat_max_len = val;
```

The type returned by **GROUP_CONCAT()** is always **VARCHAR** unless **group_concat_max_len** is greater than 512, in which case, it returns a **BLOB**.

See also **CONCAT()** and **CONCAT_WS()**: [Section 11.4, “String Functions”](#).

- **MIN([DISTINCT] *expr*), MAX([DISTINCT] *expr*)**

Returns the minimum or maximum value of *expr*. **MIN()** and **MAX()** may take a string argument; in such cases they return the minimum or maximum string value. See [Section 6.4.5, “How MySQL Uses Indexes”](#). The **DISTINCT** keyword can be used to find the minimum or maximum of the distinct values of *expr*, however, this produces the same result as omitting **DISTINCT**.

MIN() and **MAX()** return **NULL** if there were no matching rows.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
->        FROM student
```

```
-> GROUP BY student_name;
```

For `MIN()`, `MAX()`, and other aggregate functions, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set. This differs from how `ORDER BY` compares them. This is expected to be rectified in a future MySQL release.

- `STD(expr) STDDEV(expr)`

Returns the population standard deviation of *expr*. This is an extension to standard SQL. The `STDDEV()` form of this function is provided for compatibility with Oracle. The standard SQL function `STDDEV_POP()` can be used instead.

These functions return `NULL` if there were no matching rows.

- `STDDEV_POP(expr)`

Returns the population standard deviation of *expr* (the square root of `VAR_POP()`). You can also use `STD()` or `STDDEV()`, which are equivalent but not standard SQL.

`STDDEV_POP()` returns `NULL` if there were no matching rows.

- `STDDEV_SAMP(expr)`

Returns the sample standard deviation of *expr* (the square root of `VAR_SAMP()`).

`STDDEV_SAMP()` returns `NULL` if there were no matching rows.

- `SUM([DISTINCT] expr)`

Returns the sum of *expr*. If the return set has no rows, `SUM()` returns `NULL`. The `DISTINCT` keyword can be used in MySQL 5.1 to sum only the distinct values of *expr*.

`SUM()` returns `NULL` if there were no matching rows.

- `VAR_POP(expr)`

Returns the population standard variance of *expr*. It considers rows as the whole population, not as a sample, so it has the number of rows as the denominator. You can also use `VARIANCE()`, which is equivalent but is not standard SQL.

`VAR_POP()` returns `NULL` if there were no matching rows.

- `VAR_SAMP(expr)`

Returns the sample variance of *expr*. That is, the denominator is the number of rows minus one.

`VAR_SAMP()` returns `NULL` if there were no matching rows.

- `VARIANCE(expr)`

Returns the population standard variance of *expr*. This is an extension to standard SQL. The standard SQL function `VAR_POP()` can be used instead.

`VARIANCE()` returns `NULL` if there were no matching rows.

11.12.2. GROUP BY Modifiers

The `GROUP BY` clause allows a `WITH ROLLUP` modifier that causes extra rows to be added to the summary output. These rows represent higher-level (or super-aggregate) summary operations. `ROLLUP` thus allows you to answer questions at multiple levels of analysis with a single query. It can be used, for example, to provide support for OLAP (Online Analytical Processing) operations.

Suppose that a table named `sales` has `year`, `country`, `product`, and `profit` columns for recording sales profitability:

```
CREATE TABLE sales
(
  year      INT NOT NULL,
  country   VARCHAR(20) NOT NULL,
  product   VARCHAR(32) NOT NULL,
```

```
profit INT
);
```

The table's contents can be summarized per year with a simple **GROUP BY** like this:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+-----+-----+
| year | SUM(profit) |
+-----+-----+
| 2000 |          4525 |
| 2001 |          3010 |
+-----+-----+
```

This output shows the total profit for each year, but if you also want to determine the total profit summed over all years, you must add up the individual values yourself or run an additional query.

Or you can use **ROLLUP**, which provides both levels of analysis with a single query. Adding a **WITH ROLLUP** modifier to the **GROUP BY** clause causes the query to produce another row that shows the grand total over all year values:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
+-----+-----+
| year | SUM(profit) |
+-----+-----+
| 2000 |          4525 |
| 2001 |          3010 |
| NULL |          7535 |
+-----+-----+
```

The grand total super-aggregate line is identified by the value **NULL** in the **year** column.

ROLLUP has a more complex effect when there are multiple **GROUP BY** columns. In this case, each time there is a “break” (change in value) in any but the last grouping column, the query produces an extra super-aggregate summary row.

For example, without **ROLLUP**, a summary on the **sales** table based on **year**, **country**, and **product** might look like this:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product;
+-----+-----+-----+-----+
| year | country | product | SUM(profit) |
+-----+-----+-----+-----+
| 2000 | Finland | Computer |          1500 |
| 2000 | Finland | Phone   |           100 |
| 2000 | India   | Calculator |          150 |
| 2000 | India   | Computer |         1200 |
| 2000 | USA     | Calculator |           75 |
| 2000 | USA     | Computer |         1500 |
| 2001 | Finland | Phone   |           10 |
| 2001 | USA     | Calculator |           50 |
| 2001 | USA     | Computer |         2700 |
| 2001 | USA     | TV       |           250 |
+-----+-----+-----+-----+
```

The output indicates summary values only at the year/country/product level of analysis. When **ROLLUP** is added, the query produces several extra rows:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product WITH ROLLUP;
+-----+-----+-----+-----+
| year | country | product | SUM(profit) |
+-----+-----+-----+-----+
| 2000 | Finland | Computer |          1500 |
| 2000 | Finland | Phone   |           100 |
| 2000 | Finland | NULL    |          1600 |
| 2000 | India   | Calculator |          150 |
| 2000 | India   | Computer |         1200 |
| 2000 | India   | NULL    |         1350 |
| 2000 | USA     | Calculator |           75 |
| 2000 | USA     | Computer |         1500 |
| 2000 | USA     | NULL    |         1575 |
| 2000 | NULL    | NULL    |          4525 |
| 2001 | Finland | Phone   |           10 |
| 2001 | Finland | NULL    |           10 |
| 2001 | USA     | Calculator |           50 |
| 2001 | USA     | Computer |         2700 |
| 2001 | USA     | TV       |           250 |
| 2001 | USA     | NULL    |         3000 |
+-----+-----+-----+-----+
```

2001	NULL	NULL	3010
NULL	NULL	NULL	7535

For this query, adding **ROLLUP** causes the output to include summary information at four levels of analysis, not just one. Here's how to interpret the **ROLLUP** output:

- Following each set of product rows for a given year and country, an extra summary row is produced showing the total for all products. These rows have the **product** column set to **NULL**.
- Following each set of rows for a given year, an extra summary row is produced showing the total for all countries and products. These rows have the **country** and **products** columns set to **NULL**.
- Finally, following all other rows, an extra summary row is produced showing the grand total for all years, countries, and products. This row has the **year**, **country**, and **products** columns set to **NULL**.

Other Considerations When using **ROLLUP**

The following items list some behaviors specific to the MySQL implementation of **ROLLUP**:

When you use **ROLLUP**, you cannot also use an **ORDER BY** clause to sort the results. In other words, **ROLLUP** and **ORDER BY** are mutually exclusive. However, you still have some control over sort order. **GROUP BY** in MySQL sorts results, and you can use explicit **ASC** and **DESC** keywords with columns named in the **GROUP BY** list to specify sort order for individual columns. (The higher-level summary rows added by **ROLLUP** still appear after the rows from which they are calculated, regardless of the sort order.)

LIMIT can be used to restrict the number of rows returned to the client. **LIMIT** is applied after **ROLLUP**, so the limit applies against the extra rows added by **ROLLUP**. For example:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product WITH ROLLUP
-> LIMIT 5;
```

year	country	product	SUM(profit)
2000	Finland	Computer	1500
2000	Finland	Phone	100
2000	Finland	NULL	1600
2000	India	Calculator	150
2000	India	Computer	1200

Using **LIMIT** with **ROLLUP** may produce results that are more difficult to interpret, because you have less context for understanding the super-aggregate rows.

The **NULL** indicators in each super-aggregate row are produced when the row is sent to the client. The server looks at the columns named in the **GROUP BY** clause following the leftmost one that has changed value. For any column in the result set with a name that is a lexical match to any of those names, its value is set to **NULL**. (If you specify grouping columns by column number, the server identifies which columns to set to **NULL** by number.)

Because the **NULL** values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you cannot test them as **NULL** values within the query itself. For example, you cannot add **HAVING product IS NULL** to the query to eliminate from the output all but the super-aggregate rows.

On the other hand, the **NULL** values do appear as **NULL** on the client side and can be tested as such using any MySQL client programming interface.

11.12.3. **GROUP BY** and **HAVING** with Hidden Fields

MySQL extends the use of **GROUP BY** so that you can use non-aggregated columns or calculations in the **SELECT** list that do not appear in the **GROUP BY** clause. You can use this feature to get better performance by avoiding unnecessary column sorting and grouping. For example, you do not need to group on **customer.name** in the following query:

```
SELECT order.custid, customer.name, MAX(payments)
FROM order, customer
WHERE order.custid = customer.custid
GROUP BY order.custid;
```

In standard SQL, you would have to add `customer.name` to the `GROUP BY` clause. In MySQL, the name is redundant.

Do *not* use this feature if the columns you omit from the `GROUP BY` part are not constant in the group. The server is free to return any value from the group, so the results are indeterminate unless all values are the same.

A similar MySQL extension applies to the `HAVING` clause. The SQL standard does not allow the `HAVING` clause to name any column that is not found in the `GROUP BY` clause if it is not enclosed in an aggregate function. MySQL allows the use of such columns to simplify calculations. This extension assumes that the non-grouped columns will have the same group-wise values. Otherwise, the result is indeterminate.

If the `ONLY_FULL_GROUP_BY` SQL mode is enabled, the MySQL extension to `GROUP BY` does not apply. That is, columns not named in the `GROUP BY` clause cannot be used in the `SELECT` list or `HAVING` clause if not used in an aggregate function.

The select list extension also applies to `ORDER BY`. That is, you can use non-aggregated columns or calculations in the `ORDER BY` clause that do not appear in the `GROUP BY` clause. This extension does not apply if the `ONLY_FULL_GROUP_BY` SQL mode is enabled.

In some cases, you can use `MIN()` and `MAX()` to obtain a specific column value even if it isn't unique. The following gives the value of `column` from the row containing the smallest value in the `sort` column:

```
SUBSTR(MIN(CONCAT(RPAD(sort,6,' '),column)),7)
```

See [Section 3.6.4, “The Rows Holding the Group-wise Maximum of a Certain Field”](#).

Note that if you are trying to follow standard SQL, you can't use expressions in `GROUP BY` clauses. You can work around this limitation by using an alias for the expression:

```
SELECT id,FLOOR(value/100) AS val
FROM   tbl_name
GROUP BY id, val;
```

MySQL does allow expressions in `GROUP BY` clauses. For example:

```
SELECT id,FLOOR(value/100)
FROM   tbl_name
GROUP BY id, FLOOR(value/100);
```

Chapter 12. SQL Statement Syntax

This chapter describes the syntax for most of the SQL statements supported by MySQL. Additional statement descriptions can be found in the following chapters:

- The **EXPLAIN** statement is discussed in [Chapter 6, Optimization](#).
- Statements for writing stored routines are covered in [Chapter 19, Stored Procedures and Functions](#).
- Statements for writing triggers are covered in [Chapter 20, Triggers](#).
- View-related statements are covered in [Chapter 22, Views](#).
- Statements for scheduling events are covered in [Chapter 21, Event Scheduler](#).

12.1. Data Definition Statements

12.1.1. ALTER DATABASE Syntax

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification ...

alter_specification:
    [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```

ALTER DATABASE enables you to change the overall characteristics of a database. These characteristics are stored in the **db.opt** file in the database directory. To use **ALTER DATABASE**, you need the **ALTER** privilege on the database. **ALTER SCHEMA** is a synonym for **ALTER DATABASE**.

The **CHARACTER SET** clause changes the default database character set. The **COLLATE** clause changes the default database collation. [Chapter 9, Character Set Support](#), discusses character set and collation names.

The database name can be omitted, in which case the statement applies to the default database.

MySQL Enterprise

In a production environment, alteration of a database is not a common occurrence and may indicate a security breach. Advisors provided as part of the MySQL Enterprise Monitor automatically alert you when data definition statements are issued. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

12.1.2. ALTER LOGFILE GROUP Syntax

```
ALTER LOGFILE GROUP logfile_group
    ADD UNDOFILE 'file'
    [INITIAL_SIZE [=] size]
    [WAIT]
    ENGINE [=] engine
```

This statement adds an **UNDO** file named **'file'** to an existing log file group **logfile_group**. An **ALTER LOGFILE GROUP** statement has one and only one **ADD UNDOFILE** clause. No **DROP UNDOFILE** clause is supported.

The optional **INITIAL_SIZE** parameter sets the **UNDO** file's initial size in bytes; if not specified, the initial size default to **128M** (128 megabytes). You may optionally follow **size** with a one-letter abbreviation for an order of magnitude, similar to those used in **my.cnf**. Generally, this is one of the letters **M** (for megabytes) or **G** (for gigabytes).

WAIT is parsed but otherwise ignored, and so has no effect in MySQL 5.1. It is intended for future expansion.

The **ENGINE** parameter (required) determines the storage engine which is used by this log file group, with **engine** being the name of the storage engine. In MySQL 5.1, the only accepted values for **engine** are **NDB** and **NDBCLUSTER**.

Here is an example, which assumes that the log file group **lg_3** has already been created using **CREATE LOGFILE GROUP** (see [Section 12.1.8, "CREATE LOGFILE GROUP Syntax"](#)):

```
ALTER LOGFILE GROUP lg_3
ADD UNDOFILE 'undo_10.dat'
INITIAL_SIZE=32M
ENGINE=NDB;
```

When **ALTER LOGFILE GROUP** is used with **ENGINE = NDB**, an **UNDO** log file is created on each Cluster data node. You can verify that the **UNDO** files were created and obtain information about them by querying the **INFORMATION_SCHEMA.FILES** table. For example:

```
mysql> SELECT FILE_NAME, LOGFILE_GROUP_NUMBER, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE LOGFILE_GROUP_NAME = 'lg_3';
```

FILE_NAME	LOGFILE_GROUP_NUMBER	EXTRA
newdata.dat	0	CLUSTER_NODE=3
newdata.dat	0	CLUSTER_NODE=4
undo_10.dat	11	CLUSTER_NODE=3
undo_10.dat	11	CLUSTER_NODE=4

4 rows in set (0.01 sec)

(See [Section 23.21](#), “The **INFORMATION_SCHEMA.FILES** Table”.)

ALTER LOGFILE GROUP was added in MySQL 5.1.6. In MySQL 5.1, it is useful only with Disk Data storage for MySQL Cluster. See [Section 16.11](#), “MySQL Cluster Disk Data Tables”.

12.1.3. ALTER SERVER Syntax

```
ALTER SERVER server_name
OPTIONS (option ...)
```

Alters the server information for *server_name*, adjusting the specified options as per the **CREATE SERVER** command. See [Section 12.1.9](#), “**CREATE SERVER Syntax**”. The corresponding fields in the **mysql.servers** table are updated accordingly. This statement requires the **SUPER** privilege.

For example, to update the **USER** option:

```
ALTER SERVER s OPTIONS (USER 'sally');
```

ALTER SERVER does not cause an automatic commit.

12.1.4. ALTER TABLE Syntax

```
ALTER [IGNORE] TABLE tbl_name
alter_specification [, alter_specification] ...

alter_specification:
column_definition [FIRST | AFTER col_name]
| ADD [COLUMN] (column_definition,...)
| ADD {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
| ADD [CONSTRAINT [symbol]]
    PRIMARY KEY [index_type] (index_col_name,...)
| ADD [CONSTRAINT [symbol]]
    UNIQUE [INDEX|KEY] [index_name] [index_type] (index_col_name,...)
| ADD FULLTEXT [INDEX|KEY] [index_name] (index_col_name,...)
    [WITH PARSER parser_name]
| ADD SPATIAL [INDEX|KEY] [index_name] (index_col_name,...)
| ADD [CONSTRAINT [symbol]]
    FOREIGN KEY [index_name] (index_col_name,...)
    [reference_definition]
| ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
| CHANGE [COLUMN] old_col_name column_definition
    [FIRST|AFTER col_name]
| MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
| DROP [COLUMN] col_name
| DROP PRIMARY KEY
| DROP {INDEX|KEY} index_name
| DROP FOREIGN KEY fk_symbol
| DISABLE KEYS
| ENABLE KEYS
| RENAME [TO] new_tbl_name
| ORDER BY col_name [, col_name] ...
| CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
```

```

[DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
DISCARD TABLESPACE
IMPORT TABLESPACE
PARTITION BY partition_options
ADD PARTITION (partition_definition)
DROP PARTITION partition_names
COALESCE PARTITION number
REORGANIZE PARTITION partition_names INTO (partition_definitions)
ANALYZE PARTITION partition_names
CHECK PARTITION partition_names
OPTIMIZE PARTITION partition_names
REBUILD PARTITION partition_names
REPAIR PARTITION partition_names
REMOVE PARTITIONING

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

```

ALTER TABLE enables you to change the structure of an existing table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change the comment for the table and type of the table.

The syntax for many of the allowable alterations is similar to clauses of the **CREATE TABLE** statement. See [Section 12.1.10, “CREATE TABLE Syntax”](#), for more information.

Some operations may result in warnings if attempted on a table for which the storage engine does not support the operation. These warnings can be displayed with **SHOW WARNINGS**. See [Section 12.5.4.32, “SHOW WARNINGS Syntax”](#).

In most cases, **ALTER TABLE** works by making a temporary copy of the original table. The alteration is performed on the copy, and then the original table is deleted and the new one is renamed. While **ALTER TABLE** is executing, the original table is readable by other clients. Updates and writes to the table are stalled until the new table is ready, and then are automatically redirected to the new table without any failed updates.

In some cases, no temporary table is necessary:

- If you use **ALTER TABLE *tbl_name* RENAME TO *new_tbl_name*** without any other options, MySQL simply renames any files that correspond to the table *tbl_name*. (You can also use the **RENAME TABLE** statement to rename tables. See [Section 12.1.19, “RENAME TABLE Syntax”](#).) Any privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.
- **ALTER TABLE ... ADD PARTITION** creates no temporary table except for MySQL Cluster. **ADD** or **DROP** operations for **RANGE** or **LIST** partitions are immediate operations or nearly so. **ADD** or **COALESCE** operations for **HASH** or **KEY** partitions copy data between changed partitions; unless **LINEAR HASH/KEY** was used, this is much the same as creating a new table (although the operation is done partition by partition). **REORGANIZE** operations copy only changed partitions and do not touch unchanged ones.

If other cases, MySQL creates a temporary table, even if the data wouldn't strictly need to be copied (such as when you change the name of a column). For **MyISAM** tables, you can speed up the index re-creation operation (which is the slowest part of the alteration process) by setting the **myisam_sort_buffer_size** system variable to a high value.

- To use **ALTER TABLE**, you need **ALTER**, **INSERT**, and **CREATE** privileges for the table.
- **IGNORE** is a MySQL extension to standard SQL. It controls how **ALTER TABLE** works if there are duplicates on unique keys in the new table or if warnings occur when strict mode is enabled. If **IGNORE** is not specified, the copy is aborted and rolled back if duplicate-key errors occur. If **IGNORE** is specified, only the first row is used of rows with duplicates on a unique key. The other conflicting rows are deleted. Incorrect values are truncated to the closest matching acceptable value.
- *table_option* signifies a table option of the kind that can be used in the **CREATE TABLE** statement. ([Section 12.1.10, “CREATE TABLE Syntax”](#), lists all table options.) This includes options such as **ENGINE**, **AUTO_INCREMENT**, and **AVG_ROW_LENGTH**. However, **ALTER TABLE** ignores the **DATA DIRECTORY** and **INDEX DIRECTORY** table options.

For example, to convert a table to be an **InnoDB** table, use this statement:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

The outcome of attempting to change a table's storage engine is affected by whether the desired storage engine is available and the

setting of the `NO_ENGINE_SUBSTITUTION` SQL mode, as described in [Section 5.2.6, “SQL Modes”](#).

As of MySQL 5.1.11, to prevent inadvertent loss of data, `ALTER TABLE` cannot be used to change the storage engine of a table to `MERGE` or `BLACKHOLE`.

To change the value of the `AUTO_INCREMENT` counter to be used for new rows, do this:

```
ALTER TABLE t2 AUTO_INCREMENT = value;
```

You cannot reset the counter to a value less than or equal to any that have already been used. For `MyISAM`, if the value is less than or equal to the maximum value currently in the `AUTO_INCREMENT` column, the value is reset to the current maximum plus one. For `InnoDB`, *if the value is less than the current maximum value in the column, no error message is given and the current sequence value is not changed.*

- You can issue multiple `ADD`, `ALTER`, `DROP`, and `CHANGE` clauses in a single `ALTER TABLE` statement, separated by commas. This is a MySQL extension to standard SQL, which allows only one of each clause per `ALTER TABLE` statement. For example, to drop multiple columns in a single statement, do this:

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

- `CHANGE col_name`, `DROP col_name`, and `DROP INDEX` are MySQL extensions to standard SQL.
- `MODIFY` is an Oracle extension to `ALTER TABLE`.
- The word `COLUMN` is optional and can be omitted.
- `column_definition` clauses use the same syntax for `ADD` and `CHANGE` as for `CREATE TABLE`. Note that this syntax includes the column name, not just its data type. See [Section 12.1.10, “CREATE TABLE Syntax”](#).
- You can rename a column using a `CHANGE old_col_name column_definition` clause. To do so, specify the old and new column names and the type that the column currently has. For example, to rename an `INTEGER` column from `a` to `b`, you can do this:

```
ALTER TABLE t1 CHANGE a b INTEGER;
```

If you want to change a column's type but not the name, `CHANGE` syntax still requires an old and new column name, even if they are the same. For example:

```
ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

You can also use `MODIFY` to change a column's type without renaming it:

```
ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- If you use `CHANGE` or `MODIFY` to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.
- When you change a data type using `CHANGE` or `MODIFY`, MySQL tries to convert existing column values to the new type as well as possible.
- To add a column at a specific position within a table row, use `FIRST` or `AFTER col_name`. The default is to add the column last. You can also use `FIRST` and `AFTER` in `CHANGE` or `MODIFY` operations.
- `ALTER ... SET DEFAULT` or `ALTER ... DROP DEFAULT` specify a new default value for a column or remove the old default value, respectively. If the old default is removed and the column can be `NULL`, the new default is `NULL`. If the column cannot be `NULL`, MySQL assigns a default value, as described in [Section 10.1.4, “Data Type Default Values”](#).
- `DROP INDEX` removes an index. This is a MySQL extension to standard SQL. See [Section 12.1.13, “DROP INDEX Syntax”](#).
- If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well.
- If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use `DROP TABLE` in-

stead.

- **DROP PRIMARY KEY** drops the primary index.

Note

In older versions of MySQL, if no primary index existed, **DROP PRIMARY KEY** would drop the first **UNIQUE** index in the table. This is not the case in MySQL 5.1, where trying to use **DROP PRIMARY KEY** on a table with no primary key results in an error.

If you add a **UNIQUE INDEX** or **PRIMARY KEY** to a table, it is stored before any non-unique index so that MySQL can detect duplicate keys as early as possible.

- Some storage engines allow you to specify an index type when creating an index. The syntax for the *index_type* specifier is **USING type_name**. Before MySQL 5.1.10, **USING** can be given only before the index column list. As of 5.1.10, the preferred position is after the column list. Use of the option before the column list will no longer be recognized as of MySQL 5.3.
- After an **ALTER TABLE** statement, it may be necessary to run **ANALYZE TABLE** to update index cardinality information. See [Section 12.5.4.18, “SHOW INDEX Syntax”](#).
- **ORDER BY** enables you to create the new table with the rows in a specific order. Note that the table does not remain in this order after inserts and deletes. This option is useful primarily when you know that you are mostly to query the rows in a certain order most of the time. By using this option after major changes to the table, you might be able to get higher performance. In some cases, it might make sorting easier for MySQL if the table is in order by the column that you want to order it by later.

ORDER BY syntax allows for one or more column names to be specified for sorting, each of which optionally can be followed by **ASC** or **DESC** to indicate ascending or descending sort order, respectively. The default is ascending order. Only column names are allowed as sort criteria; arbitrary expressions are not allowed.

- If you use **ALTER TABLE** on a **MyISAM** table, all non-unique indexes are created in a separate batch (as for **REPAIR TABLE**). This should make **ALTER TABLE** much faster when you have many indexes.

This feature can be activated explicitly. **ALTER TABLE ... DISABLE KEYS** tells MySQL to stop updating non-unique indexes for a **MyISAM** table. **ALTER TABLE ... ENABLE KEYS** then should be used to re-create missing indexes. MySQL does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should give a considerable speedup. Using **ALTER TABLE ... DISABLE KEYS** requires the **INDEX** privilege in addition to the privileges mentioned earlier.

While the non-unique indexes are disabled, they are ignored for statements such as **SELECT** and **EXPLAIN** that otherwise would use them.

ENABLE KEYS and **DISABLE KEYS** were not supported for partitioned tables prior to MySQL 5.1.11. ([Bug#19502](#))

- If **ALTER TABLE** for an **InnoDB** table results in changes to column values (for example, because a column is truncated), **InnoDB**'s **FOREIGN KEY** constraint checks do not notice possible violations caused by changing the values.
- The **FOREIGN KEY** and **REFERENCES** clauses are supported by the **InnoDB** storage engine, which implements **ADD [CONSTRAINT [symbol]] FOREIGN KEY (...) REFERENCES ... (...)**. See [Section 13.5.6.4, “FOREIGN KEY Constraints”](#). For other storage engines, the clauses are parsed but ignored. The **CHECK** clause is parsed but ignored by all storage engines. See [Section 12.1.10, “CREATE TABLE Syntax”](#). The reason for accepting but ignoring syntax clauses is for compatibility, to make it easier to port code from other SQL servers, and to run applications that create tables with references. See [Section 1.9.5, “MySQL Differences from Standard SQL”](#).

You cannot add a foreign key and drop a foreign key in separate clauses of a single **ALTER TABLE** statement. You must use separate statements.

Important

The inline **REFERENCES** specifications where the references are defined as part of the column specification are silently ignored by **InnoDB**. **InnoDB** only accepts **REFERENCES** clauses when specified as part of a separate **FOREIGN KEY** specification.

- **InnoDB** supports the use of **ALTER TABLE** to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

You cannot add a foreign key and drop a foreign key in separate clauses of a single **ALTER TABLE** statement. You must use separate statements.

For more information, see [Section 13.5.6.4, “FOREIGN KEY Constraints”](#).

- Pending **INSERT DELAYED** statements are lost if a table is write locked and **ALTER TABLE** is used to modify the table structure.
- If you want to change the table default character set and all character columns (**CHAR**, **VARCHAR**, **TEXT**) to a new character set, use a statement like this:

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
```

Warning

The preceding operation converts column values between the character sets. This is *not* what you want if you have a column in one character set (like **latin1**) but the stored values actually use some other, incompatible character set (like **utf8**). In this case, you have to do the following for each such column:

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

The reason this works is that there is no conversion when you convert to or from **BLOB** columns.

If you specify **CONVERT TO CHARACTER SET binary**, the **CHAR**, **VARCHAR**, and **TEXT** columns are converted to their corresponding binary string types (**BINARY**, **VARBINARY**, **BLOB**). This means that the columns no longer will have a character set and a subsequent **CONVERT TO** operation will not apply to them.

To change only the *default* character set for a table, use this statement:

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

The word **DEFAULT** is optional. The default character set is the character set that is used if you do not specify the character set for a new column which you add to a table (for example, with **ALTER TABLE ... ADD column**).

- For an **InnoDB** table that is created with its own tablespace in an **.ibd** file, that file can be discarded and imported. To discard the **.ibd** file, use this statement:

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

This deletes the current **.ibd** file, so be sure that you have a backup first. Attempting to access the table while the tablespace file is discarded results in an error.

To import the backup **.ibd** file back into the table, copy it into the database directory, and then issue this statement:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

See [Section 13.5.3.1, “Using Per-Table Tablespaces”](#).

- A number of partitioning-related extensions to **ALTER TABLE** were added in MySQL 5.1.5. These can be used with partitioned tables for repartitioning, for adding, dropping, merging, and splitting partitions, and for performing partitioning maintenance.

Simply using a *partition_options* clause with **ALTER TABLE** on a partitioned table repartitions the table according to the partitioning scheme defined by the *partition_options*. This clause always begins with **PARTITION BY**, and follows the same syntax and other rules as apply to the *partition_options* clause for **CREATE TABLE** (see [Section 12.1.10, “CREATE TABLE Syntax”](#), for more detailed information), and can also be used to partition an existing table that is not already partitioned. For example, consider a (non-partitioned) table defined as shown here:

```
CREATE TABLE t1 (
  id INT,
  year_col INT
);
```

This table can be partitioned by **HASH**, using the **id** column as the partitioning key, into 8 partitions by means of this statement:

```
ALTER TABLE t1
PARTITION BY HASH(id)
PARTITIONS 8;
```

The table that results from using an `ALTER TABLE ... PARTITION BY` statement must follow the same rules as one created using `CREATE TABLE ... PARTITION BY`. This includes the rules governing the relationship between any unique keys (including any primary key) that the table might have, and the column or columns used in the partitioning expression, as discussed in [Partitioning Limitations: Partitioning Keys and Unique Keys](#). The `CREATE TABLE ... PARTITION BY` rules for specifying the number of partitions also apply to `ALTER TABLE ... PARTITION BY`.

`ALTER TABLE ... PARTITION BY` became available in MySQL 5.1.6.

The *partition_definition* clause for `ALTER TABLE ADD PARTITION` supports the same options as the clause of the same name does for the `CREATE TABLE` statement clause of the same name. (See [Section 12.1.10, “CREATE TABLE Syntax”](#), for the syntax and description.) Suppose that you have the partitioned table created as shown here:

```
CREATE TABLE t1 (
  id INT,
  year_col INT
)
PARTITION BY RANGE (year_col) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1995),
  PARTITION p2 VALUES LESS THAN (1999)
);
```

You can add a new partition `p3` to this table for storing values less than `2002` as follows:

```
ALTER TABLE t1 ADD PARTITION (PARTITION p3 VALUES LESS THAN (2002));
```

`DROP PARTITION` can be used to drop one or more `RANGE` or `LIST` partitions. This statement cannot be used with `HASH` or `KEY` partitions; instead, use `COALESCE PARTITION` (see below). Any data that was stored in the dropped partitions named in the *partition_names* list is discarded. For example, given the table `t1` defined previously, you can drop the partitions named `p0` and `p1` as shown here:

```
ALTER TABLE t1 DROP PARTITION p0, p1;
```

Note that `DROP PARTITION` does not work with tables that use the `NDB Cluster` storage engine. See [Section 17.3.1, “Management of RANGE and LIST Partitions”](#), and [Section 16.13, “Known Limitations of MySQL Cluster”](#).

`ADD PARTITION` and `DROP PARTITION` do not currently support `IF [NOT] EXISTS`. It is also not possible to rename a partition or a partitioned table. Instead, if you wish to rename a partition, you must drop and re-create the partition; if you wish to rename a partitioned table, you must instead drop all partitions, rename the table, and then add back the partitions that were dropped.

`COALESCE PARTITION` can be used with a table that is partitioned by `HASH` or `KEY` to reduce the number of partitions by *number*. Suppose that you have created table `t2` using the following definition:

```
CREATE TABLE t2 (
  name VARCHAR (30),
  started DATE
)
PARTITION BY HASH( YEAR(started) )
PARTITIONS 6;
```

You can reduce the number of partitions used by `t2` from 6 to 4 using the following statement:

```
ALTER TABLE t2 COALESCE PARTITION 2;
```

The data contained in the last *number* partitions will be merged into the remaining partitions. In this case, partitions 4 and 5 will be merged into the first 4 partitions (the partitions numbered 0, 1, 2, and 3).

To change some but not all the partitions used by a partitioned table, you can use `REORGANIZE PARTITION`. This statement can be used in several ways:

- To merge a set of partitions into a single partition. This can be done by naming several partitions in the *partition_names* list and supplying a single definition for *partition_definition*.

- To split an existing partition into several partitions. You can accomplish this by naming a single partition for *partition_names* and providing multiple *partition_definitions*.
- To change the ranges for a subset of partitions defined using **VALUES LESS THAN** or the value lists for a subset of partitions defined using **VALUES IN**.

Note

For partitions that have not been explicitly named, MySQL automatically provides the default names **p0**, **p1**, **p2**, and so on. As of MySQL 5.1.7, the same is true with regard to subpartitions.

For more detailed information about and examples of **ALTER TABLE ... REORGANIZE PARTITION** statements, see [Section 17.3, “Partition Management”](#).

- Several additional clauses provide partition maintenance and repair functionality analogous to that implemented for non-partitioned tables by statements such as **CHECK TABLE** and **REPAIR TABLE** (which are *not* supported for partitioned tables). These include **ANALYZE PARTITION**, **CHECK PARTITION**, **OPTIMIZE PARTITION**, **REBUILD PARTITION**, and **REPAIR PARTITION**. Each of these options takes a *partition_names* clause consisting of one or more names of partitions, separated by commas. The partitions must already exist in the table to be altered. For more information, and for examples of these, see [Section 17.3.3, “Maintenance of Partitions”](#).
- **REMOVE PARTITIONING** was introduced in MySQL 5.1.8 for the purpose of removing a table's partitioning without otherwise affecting the table or its data. (Previously, this was done using the **ENGINE** option.) This option can be combined with other **ALTER TABLE** options such as those used to add, drop, or rename drop columns or indexes.

In MySQL 5.1.7 and earlier, using the **ENGINE** option with **ALTER TABLE** caused any partitioning that a table might have had to be removed. Beginning with MySQL 5.1.8, this option merely changes the storage engine used by the table and no longer affects partitioning in any way.

With the `mysql_info()` C API function, you can find out how many rows were copied, and (when **IGNORE** is used) how many rows were deleted due to duplication of unique key values. See [Section 25.2.3.35, “mysql_info\(\)”](#).

Here are some examples that show uses of **ALTER TABLE**. Begin with a table **t1** that is created as shown here:

```
CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

To rename the table from **t1** to **t2**:

```
ALTER TABLE t1 RENAME t2;
```

To change column **a** from **INTEGER** to **TINYINT NOT NULL** (leaving the name the same), and to change column **b** from **CHAR(10)** to **CHAR(20)** as well as renaming it from **b** to **c**:

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new **TIMESTAMP** column named **d**:

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

To add indexes on column **d** and on column **a**:

```
ALTER TABLE t2 ADD INDEX (d), ADD INDEX (a);
```

To remove column **c**:

```
ALTER TABLE t2 DROP COLUMN c;
```

To add a new **AUTO_INCREMENT** integer column named **c**:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,  
ADD PRIMARY KEY (c);
```

Note that we indexed `c` (as a [PRIMARY KEY](#)), because [AUTO_INCREMENT](#) columns must be indexed, and also that we declare `c` as [NOT NULL](#), because primary key columns cannot be [NULL](#).

When you add an [AUTO_INCREMENT](#) column, column values are filled in with sequence numbers for you automatically. For [MyISAM](#) tables, you can set the first sequence number by executing [SET INSERT_ID=value](#) before [ALTER TABLE](#) or by using the [AUTO_INCREMENT=value](#) table option. See [Section 12.5.3, “SET Syntax”](#).

With [MyISAM](#) tables, if you do not change the [AUTO_INCREMENT](#) column, the sequence number is not affected. If you drop an [AUTO_INCREMENT](#) column and then add another [AUTO_INCREMENT](#) column, the numbers are resequenced beginning with 1.

When replication is used, adding an [AUTO_INCREMENT](#) column to a table might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an [AUTO_INCREMENT](#) number. Assuming that you want to add an [AUTO_INCREMENT](#) column to the table `t1`, the following statements produce a new table `t2` identical to `t1` but with an [AUTO_INCREMENT](#) column:

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

This assumes that the table `t1` has columns `col1` and `col2`.

This set of statements will also produce a new table `t2` identical to `t1`, with the addition of an [AUTO_INCREMENT](#) column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important

To guarantee the same ordering on both master and slave, *all* columns of `t1` must be referenced in the [ORDER BY](#) clause.

Regardless of the method used to create and populate the copy having the [AUTO_INCREMENT](#) column, the final step is to drop the original table and then rename the copy:

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

See also [Section B.1.7.1, “Problems with ALTER TABLE”](#).

12.1.5. ALTER TABLESPACE Syntax

```
ALTER TABLESPACE tablespace
{ADD|DROP} DATAFILE 'file'
[INITIAL_SIZE [=] size]
[WAIT]
ENGINE [=] engine
```

This statement can be used either to add a new data file, or to drop a data file from a tablespace.

The [ADD DATAFILE](#) variant allows you to specify an initial size using an [INITIAL_SIZE](#) clause, where *size* is measured in bytes; the default value is [128M](#) (128 megabytes). You may optionally follow this integer value with a one-letter abbreviation for an order of magnitude, similar to those used in [my.cnf](#). Generally, this is one of the letters [M](#) (for megabytes) or [G](#) (for gigabytes).

Once a data file has been created, its size cannot be changed; however, you can add more data files to the tablespace using additional [ALTER TABLESPACE ... ADD DATAFILE](#) statements.

Using [DROP DATAFILE](#) with [ALTER TABLESPACE](#) drops the data file '*file*' from the tablespace. This file must already have been added to the tablespace using [CREATE TABLESPACE](#) or [ALTER TABLESPACE](#); otherwise an error will result.

Both [ALTER TABLESPACE ... ADD DATAFILE](#) and [ALTER TABLESPACE ... DROP DATAFILE](#) require an [ENGINE](#) clause which specifies the storage engine used by the tablespace. In MySQL 5.1, the only accepted values for *engine* are [NDB](#) and [NDBCLUSTER](#).

[WAIT](#) is parsed but otherwise ignored, and so has no effect in MySQL 5.1. It is intended for future expansion.

When [ALTER TABLESPACE ... ADD DATAFILE](#) is used with [ENGINE = NDB](#), a data file is created on each Cluster data node.

You can verify that the data files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example, the following query shows all data files belonging to the tablespace named `newts`:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_NAME, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE TABLESPACE_NAME = 'newts' AND FILE_TYPE = 'DATAFILE';
```

LOGFILE_GROUP_NAME	FILE_NAME	EXTRA
lg_3	newdata.dat	CLUSTER_NODE=3
lg_3	newdata.dat	CLUSTER_NODE=4
lg_3	newdata2.dat	CLUSTER_NODE=3
lg_3	newdata2.dat	CLUSTER_NODE=4

2 rows in set (0.03 sec)

See [Section 23.21](#), “The `INFORMATION_SCHEMA.FILES` Table”.

`ALTER TABLESPACE` was added in MySQL 5.1.6. In MySQL 5.1, it is useful only with Disk Data storage for MySQL Cluster. See [Section 16.11](#), “MySQL Cluster Disk Data Tables”.

12.1.6. CREATE DATABASE Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
[create_specification ...]
```

```
create_specification:
[DEFAULT] CHARACTER SET charset_name
| [DEFAULT] COLLATE collation_name
```

`CREATE DATABASE` creates a database with the given name. To use this statement, you need the `CREATE` privilege for the database. `CREATE SCHEMA` is a synonym for `CREATE DATABASE`.

An error occurs if the database exists and you did not specify `IF NOT EXISTS`.

`create_specification` options specify database characteristics. Database characteristics are stored in the `db.opt` file in the database directory. The `CHARACTER SET` clause specifies the default database character set. The `COLLATE` clause specifies the default database collation. [Chapter 9, Character Set Support](#), discusses character set and collation names.

A database in MySQL is implemented as a directory containing files that correspond to tables in the database. Because there are no tables in a database when it is initially created, the `CREATE DATABASE` statement creates only a directory under the MySQL data directory and the `db.opt` file. Rules for allowable database names are given in [Section 8.2](#), “Identifiers”. If a database name contains special characters, the name for the database directory contains encoded versions of those characters as described in [Section 8.2.3](#), “Mapping of Identifiers to Filenames”.

If you manually create a directory under the data directory (for example, with `mkdir`), the server considers it a database directory and it shows up in the output of `SHOW DATABASES`.

You can also use the `mysqladmin` program to create databases. See [Section 7.10](#), “`mysqladmin` — Client for Administering a MySQL Server”.

12.1.7. CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
[index_type]
ON tbl_name (index_col_name,...)
[index_option ...]
```

```
index_col_name:
col_name [(length)] [ASC | DESC]
```

```
index_type:
USING {BTREE | HASH | RTREE}
```

```
index_option:
KEY_BLOCK_SIZE value
| index_type
| WITH PARSER parser_name
```

`CREATE INDEX` is mapped to an `ALTER TABLE` statement to create indexes. See [Section 12.1.4](#), “`ALTER TABLE` Syntax”. `CREATE INDEX` cannot be used to create a `PRIMARY KEY`; use `ALTER TABLE` instead. For more information about indexes, see [Section 6.4.5](#), “How MySQL Uses Indexes”.

Normally, you create all indexes on a table at the time the table itself is created with `CREATE TABLE`. See [Section 12.1.10, “CREATE TABLE Syntax”](#). `CREATE INDEX` enables you to add indexes to existing tables.

A column list of the form `(col1,col2,...)` creates a multiple-column index. Index values are formed by concatenating the values of the given columns.

Indexes can be created that use only the leading part of column values, using `col_name(length)` syntax to specify an index prefix length:

- Prefixes can be specified for `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` columns.
- `BLOB` and `TEXT` columns also can be indexed, but a prefix length *must* be given.
- Prefix lengths are given in characters for non-binary string types and in bytes for binary string types. That is, index entries consist of the first *length* characters of each column value for `CHAR`, `VARCHAR`, and `TEXT` columns, and the first *length* bytes of each column value for `BINARY`, `VARBINARY`, and `BLOB` columns.
- For spatial columns, prefix values can be given as described later in this section.

The statement shown here creates an index using the first 10 characters of the `name` column:

```
CREATE INDEX part_of_name ON customer (name(10));
```

If names in the column usually differ in the first 10 characters, this index should not be much slower than an index created from the entire `name` column. Also, using column prefixes for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up `INSERT` operations.

Prefix lengths are storage engine-dependent (for example, a prefix can be up to 1000 bytes long for `MyISAM` tables, 767 bytes for `InnoDB` tables). Note that prefix limits are measured in bytes, whereas the prefix length in `CREATE INDEX` statements is interpreted as number of characters for non-binary data types (`CHAR`, `VARCHAR`, `TEXT`). Take this into account when specifying a prefix length for a column that uses a multi-byte character set. For example, `utf8` columns require up to three index bytes per character.

A `UNIQUE` index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. For all engines, a `UNIQUE` index allows multiple `NULL` values for columns that can contain `NULL`. If you specify a prefix value for a column in a `UNIQUE` index, the column values must be unique within the prefix.

MySQL Enterprise

Lack of proper indexes can greatly reduce performance. Subscribe to the MySQL Enterprise Monitor for notification of inefficient use of indexes. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

`FULLTEXT` indexes are supported only for `MyISAM` tables and can include only `CHAR`, `VARCHAR`, and `TEXT` columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See [Section 11.8, “Full-Text Search Functions”](#), for details of operation.

The `MyISAM`, `InnoDB`, `NDB`, `BDB`, and `ARCHIVE` storage engines support spatial columns such as (`POINT` and `GEOMETRY`. ([Chapter 18, Spatial Extensions](#), describes the spatial data types.) However, support for spatial column indexing varies among engines. Spatial and non-spatial indexes are available according to the following rules.

Spatial indexes (created using `SPATIAL INDEX`):

- Available only for `MyISAM` tables. Specifying a `SPATIAL INDEX` for other storage engines results in an error.
- Indexed columns must be `NOT NULL`.
- In MySQL 5.1, column prefix lengths are prohibited. The full width of each column is indexed.

Non-spatial indexes (created with `INDEX`, `UNIQUE`, or `PRIMARY KEY`):

- Allowed for any storage engine that supports spatial columns except `ARCHIVE`.

- Columns can be **NULL** unless the index is a primary key.
- For each spatial column in a non-**SPATIAL** index except **POINT** columns, a column prefix length must be specified. (This is the same requirement as for indexed **BLOB** columns.) The prefix length is given in bytes.
- The index type for a non-**SPATIAL** index depends on the storage engine. Currently, B-tree is used.

In MySQL 5.1:

- You can add an index on a column that can have **NULL** values only if you are using the **MyISAM**, **InnoDB**, or **MEMORY** storage engine.
- You can add an index on a **BLOB** or **TEXT** column only if you are using the **MyISAM**, or **InnoDB** storage engine.

An *index_col_name* specification can end with **ASC** or **DESC**. These keywords are allowed for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

Following the index column list, index options can be given. An *index_option* value can be any of the following:

- KEY_BLOCK_SIZE** *value*

This option provides a hint to the storage engine about the size to use for index key blocks. The engine is allowed to change the value if necessary. A value of 0 indicates that the default value should be used. **KEY_BLOCK_SIZE** was added in MySQL 5.1.10.

- index_type*

Some storage engines allow you to specify an index type when creating an index. The allowable index type values supported by different storage engines are shown in the following table. Where multiple index types are listed, the first one is the default when no index type specifier is given.

Storage Engine	Allowable Index Types
MyISAM	BTREE , RTREE
InnoDB	BTREE
MEMORY/HEAP	HASH , BTREE
NDB	HASH

The **RTREE** index type is allowable only for **SPATIAL** indexes.

If you specify an index type that is not legal for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.

Examples:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index USING BTREE ON lookup (id);
```

For indexes on **NDB** table columns, the **USING** clause can be specified only for a unique index or primary key. In such cases, the **USING HASH** clause prevents the creation of an implicit ordered index. Without **USING HASH**, a statement defining a unique index or primary key automatically results in the creation of a **HASH** index in addition to the ordered index, both of which index the same set of columns.

TYPE type_name is recognized as a synonym for **USING type_name**. However, **USING** is the preferred form.

Note: Before MySQL 5.1.10, this option can be given only before the **ON tbl_name** clause. Use of the option in this position is deprecated as of 5.1.10; support for it is to be dropped in a future MySQL release.

- WITH PARSER** *parser_name*

This option can be used only with **FULLTEXT** indexes. It associates a parser plugin with the index if full-text indexing and search-

ing operations need special handling. See [Section 28.2, “The MySQL Plugin Interface”](#), for details on creating plugins.

12.1.8. CREATE LOGFILE GROUP Syntax

```
CREATE LOGFILE GROUP logfile_group
  ADD UNDOFILE 'undo_file'
  [INITIAL_SIZE [=] initial_size]
  [UNDO_BUFFER_SIZE [=] undo_buffer_size]
  [REDO_BUFFER_SIZE [=] redo_buffer_size]
  [NODEGROUP [=] nodegroup_id]
  [WAIT]
  [COMMENT [=] comment_text]
  [ENGINE [=] engine_name]
```

This statement creates a new log file group named *logfile_group* having a single **UNDO** file named '*undo_file*'. A **CREATE LOGFILE GROUP** statement has one and only one **ADD UNDOFILE** clause.

Beginning with MySQL 5.1.8, you can have only one log file group per Cluster at any given time. (See [Bug#16386](#))

The optional **INITIAL_SIZE** parameter sets the **UNDO** file's initial size; if not specified, it defaults to **128M** (128 megabytes). The optional **UNDO_BUFFER_SIZE** parameter sets the size used by the **UNDO** buffer for the log file group; The default value for **UNDO_BUFFER_SIZE** is **8M** (eight megabytes); this value cannot exceed the amount of system memory available. Both of these parameters are specified in bytes. You may optionally follow either or both of these with a one-letter abbreviation for an order of magnitude, similar to those used in *my.cnf*. Generally, this is one of the letters **M** (for megabytes) or **G** (for gigabytes).

The **ENGINE** parameter determines the storage engine to be used by this log file group, with *engine* being the name of the storage engine. In MySQL 5.1, *engine* must be one of the values **NDB** or **NDBCLUSTER**.

REDO_BUFFER_SIZE, **NODEGROUP**, **WAIT**, and **COMMENT** are parsed but ignored, and so have no effect in MySQL 5.1. These options are intended for future expansion.

When used with **ENGINE [=] NDB**, a log file group and associated **UNDO** log file are created on each Cluster data node. You can verify that the **UNDO** files were created and obtain information about them by querying the **INFORMATION_SCHEMA.FILES** table. For example:

```
mysql> SELECT LOGFILE_GROUP_NAME, LOGFILE_GROUP_NUMBER, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE FILE_NAME = 'undo_10.dat';
+-----+-----+-----+
| LOGFILE_GROUP_NAME | LOGFILE_GROUP_NUMBER | EXTRA          |
+-----+-----+-----+
| lg_3               | 11                   | CLUSTER_NODE=3 |
| lg_3               | 11                   | CLUSTER_NODE=4 |
+-----+-----+-----+
2 rows in set (0.06 sec)
```

(See [Section 23.21, “The INFORMATION_SCHEMA FILES Table”](#).)

CREATE LOGFILE GROUP was added in MySQL 5.1.6. In MySQL 5.1, it is useful only with Disk Data storage for MySQL Cluster. See [Section 16.11, “MySQL Cluster Disk Data Tables”](#).

12.1.9. CREATE SERVER Syntax

```
CREATE SERVER server_name
  FOREIGN DATA WRAPPER wrapper_name
  OPTIONS (option ...)

option:
{
  HOST character-literal
  DATABASE character-literal
  USER character-literal
  PASSWORD character-literal
  SOCKET character-literal
  OWNER character-literal
  PORT numeric-literal }
```

This statement creates the definition of a server for use with the **FEDERATED** storage engine. The **CREATE SERVER** statement creates a new row within the **servers** table within the **mysql** database. This statement requires the **SUPER** privilege.

The *server_name* should be a unique reference to the server. Server definitions are global within the scope of the server, it is not

possible to qualify the server definition to a specific database. *server_name* has a maximum length of 63 characters (names longer than 63 characters are silently truncated), and is case insensitive. You may specify the name using single quotes.

The *wrapper_name* should be `mysql`, and may be quoted with single quotes. Other values for *wrapper_name* are not currently supported.

For each *option* you must specify either a character literal or numeric literal. Character literals are UTF8, support a maximum length of 64 characters and default to a blank (empty) string. String literals are silently truncated to 64 characters. Numeric literals must be a number between 0 and 9999, default value is 0.

The `CREATE SERVER` statement creates an entry in the `mysql.server` table that can later be used with the `CREATE TABLE` statement when creating a `FEDERATED` table. The options that you specify will be used to populate the columns in the `mysql.server` table. The table columns are `Server_name`, `Host`, `Db`, `Username`, `Password`, `Port` and `Socket`.

For example:

```
CREATE SERVER s
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
```

The data stored in the table can be used when creating a connection to a `FEDERATED` table:

```
CREATE TABLE t (s1 INT) ENGINE=FEDERATED CONNECTION='s';
```

For more information, see [Section 13.9, “The FEDERATED Storage Engine”](#).

`CREATE SERVER` does not cause an automatic commit.

12.1.10. CREATE TABLE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
(create_definition,...)
[table_option ...]
[partition_options]
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
[(create_definition,...)]
[table_option ...]
[PARTITION BY partition_options]
select_statement
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
{ LIKE old_tbl_name | (LIKE old_tbl_name) }

create_definition:
  column_definition
| [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
  [index_option ...]
| {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
  [index_option ...]
| [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
  [index_name] [index_type] (index_col_name,...)
  [index_option ...]
| {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...)
  [index_option ...]
| [CONSTRAINT [symbol]] FOREIGN KEY
  [index_name] (index_col_name,...) [reference_definition]
| CHECK (expr)

column_definition:
  col_name data_type [NOT NULL | NULL] [DEFAULT default_value]
  [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
  [COMMENT 'string'] [reference_definition]
  [COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
  [STORAGE {DISK|MEMORY}]

data_type:
  BIT[(length)]
| TINYINT[(length)] [UNSIGNED] [ZEROFILL]
| SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
| MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
```

```

| INT[(length)] [UNSIGNED] [ZEROFILL]
| INTEGER[(length)] [UNSIGNED] [ZEROFILL]
| BIGINT[(length)] [UNSIGNED] [ZEROFILL]
| REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
| DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
| FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
| DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
| NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
| DATE
| TIME
| TIMESTAMP
| DATETIME
| YEAR
| CHAR(length)
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| VARCHAR(length)
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| BINARY(length)
| VARBINARY(length)
| TINYBLOB
| BLOB
| MEDIUMBLOB
| LONGBLOB
| TINYTEXT [BINARY]
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| TEXT [BINARY]
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| MEDIUMTEXT [BINARY]
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| LONGTEXT [BINARY]
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| ENUM(value1,value2,value3,...)
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| SET(value1,value2,value3,...)
|   [CHARACTER SET charset_name] [COLLATE collation_name]
| spatial_type

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    KEY_BLOCK_SIZE value
| index_type
| WITH PARSE parser_name

reference_definition:
    REFERENCES tbl_name [(index_col_name,...)]
        [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
        [ON DELETE reference_option]
        [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION

table_option:
    [TABLESPACE tablespace_name STORAGE DISK]
    ENGINE [=] engine_name
    AUTO_INCREMENT [=] value
    AVG_ROW_LENGTH [=] value
    [DEFAULT] CHARACTER SET charset_name
    CHECKSUM [=] {0 | 1}
    COLLATE collation_name
    COMMENT [=] 'string'
    CONNECTION [=] 'connect_string'
    DATA DIRECTORY [=] 'absolute path to directory'
    DELAY_KEY_WRITE [=] {0 | 1}
    INDEX DIRECTORY [=] 'absolute path to directory'
    INSERT_METHOD [=] { NO | FIRST | LAST }
    KEY_BLOCK_SIZE [=] value
    MAX_ROWS [=] value
    MIN_ROWS [=] value
    PACK_KEYS [=] {0 | 1 | DEFAULT}
    PASSWORD [=] 'string'
    ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
    UNION [=] (tbl_name[,tbl_name]...)

partition_options:
    [LINEAR] HASH(expr)
    | [LINEAR] KEY(column_list)
    | RANGE(expr)
    | LIST(expr)
    [PARTITIONS num]
    [SUBPARTITION BY
        [LINEAR] HASH(expr)
        | [LINEAR] KEY(column_list)
        [SUBPARTITIONS num]
    ]
    [(partition_definition [, partition_definition] ...)]

```

```

partition_definition:
PARTITION partition_name
[VALUES {LESS THAN (expr) | MAXVALUE | IN (value_list)}]
[[STORAGE] ENGINE [=] engine_name]
[COMMENT [=] 'comment_text']
[DATA DIRECTORY [=] 'data_dir']
[INDEX DIRECTORY [=] 'index_dir']
[MAX_ROWS [=] max_number_of_rows]
[MIN_ROWS [=] min_number_of_rows]
[TABLESPACE [=] (tablespace_name)]
[NOGROUP [=] node_group_id]
[(subpartition_definition [, subpartition_definition] ...)]

subpartition_definition:
SUBPARTITION logical_name
[[STORAGE] ENGINE [=] engine_name]
[COMMENT [=] 'comment_text']
[DATA DIRECTORY [=] 'data_dir']
[INDEX DIRECTORY [=] 'index_dir']
[MAX_ROWS [=] max_number_of_rows]
[MIN_ROWS [=] min_number_of_rows]
[TABLESPACE [=] (tablespace_name)]
[NOGROUP [=] node_group_id]

select_statement:
[IGNORE | REPLACE] [AS] SELECT ... (Some legal select statement)

```

CREATE TABLE creates a table with the given name. You must have the **CREATE** privilege for the table.

Rules for allowable table names are given in [Section 8.2, “Identifiers”](#). By default, the table is created in the default database. An error occurs if the table exists, if there is no default database, or if the database does not exist.

The table name can be specified as *db_name.tbl_name* to create the table in a specific database. This works regardless of whether there is a default database, assuming that the database exists. If you use quoted identifiers, quote the database and table names separately. For example, write ``mydb`.`mytbl``, not ``mydb.mytbl``.

You can use the **TEMPORARY** keyword when creating a table. A **TEMPORARY** table is visible only to the current connection, and is dropped automatically when the connection is closed. This means that two different connections can use the same temporary table name without conflicting with each other or with an existing non-**TEMPORARY** table of the same name. (The existing table is hidden until the temporary table is dropped.) To create temporary tables, you must have the **CREATE TEMPORARY TABLES** privilege.

Note

CREATE TABLE does not automatically commit the current active transaction if you use the **TEMPORARY** keyword.

The keywords **IF NOT EXISTS** prevent an error from occurring if the table exists. However, there is no verification that the existing table has a structure identical to that indicated by the **CREATE TABLE** statement.

Note

If you use **IF NOT EXISTS** in a **CREATE TABLE ... SELECT** statement, any rows selected by the **SELECT** part are inserted regardless of whether the table already exists.

MySQL represents each table by an `.frm` table format (definition) file in the database directory. The storage engine for the table might create other files as well. In the case of **MyISAM** tables, the storage engine creates data and index files. Thus, for each **MyISAM** table *tbl_name*, there are three disk files:

File	Purpose
<i>tbl_name.frm</i>	Table format (definition) file
<i>tbl_name.MYD</i>	Data file
<i>tbl_name.MYI</i>	Index file

[Chapter 13, Storage Engines](#), describes what files each storage engine creates to represent tables. If a table name contains special characters, the names for the table files contain encoded versions of those characters as described in [Section 8.2.3, “Mapping of Identifiers to Filenames”](#).

data_type represents the data type is a column definition. *spatial_type* represents a spatial data type. The data type syntax shown is representative only. For a full description of the syntax available for specifying column data types, as well as information about the properties of each type, see [Chapter 10, Data Types](#), and [Chapter 18, Spatial Extensions](#).

Some attributes do not apply to all data types. **AUTO_INCREMENT** applies only to integer types. **DEFAULT** does not apply to the **BLOB** or **TEXT** types.

- If neither **NULL** nor **NOT NULL** is specified, the column is treated as though **NULL** had been specified.
- An integer column can have the additional attribute **AUTO_INCREMENT**. When you insert a value of **NULL** (recommended) or **0** into an indexed **AUTO_INCREMENT** column, the column is set to the next sequence value. Typically this is *value*+1, where *value* is the largest value for the column currently in the table. **AUTO_INCREMENT** sequences begin with 1.

To retrieve an **AUTO_INCREMENT** value after inserting a row, use the **LAST_INSERT_ID()** SQL function or the **mysql_insert_id()** C API function. See [Section 11.11.3, “Information Functions”](#), and [Section 25.2.3.37, “mysql_insert_id\(\)”](#).

If the **NO_AUTO_VALUE_ON_ZERO** SQL mode is enabled, you can store **0** in **AUTO_INCREMENT** columns as **0** without generating a new sequence value. See [Section 5.2.6, “SQL Modes”](#).

Note

There can be only one **AUTO_INCREMENT** column per table, it must be indexed, and it cannot have a **DEFAULT** value. An **AUTO_INCREMENT** column works properly only if it contains only positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to avoid precision problems when numbers “wrap” over from positive to negative and also to ensure that you do not accidentally get an **AUTO_INCREMENT** column that contains **0**.

For **MyISAM** tables, you can specify an **AUTO_INCREMENT** secondary column in a multiple-column key. See [Section 3.6.9, “Using AUTO_INCREMENT”](#).

To make MySQL compatible with some ODBC applications, you can find the **AUTO_INCREMENT** value for the last inserted row with the following query:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

For information about **InnoDB** and **AUTO_INCREMENT**, see [Section 13.5.6.3, “How AUTO_INCREMENT Handling Works in InnoDB”](#).

- Character data types (**CHAR**, **VARCHAR**, **TEXT**) can include **CHARACTER SET** and **COLLATE** attributes to specify the character set and collation for the column. For details, see [Chapter 9, Character Set Support](#). **CHARSET** is a synonym for **CHARACTER SET**. Example:

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

MySQL 5.1 interprets length specifications in character column definitions in characters. (Versions before MySQL 4.1 interpreted them in bytes.) Lengths for **BINARY** and **VARBINARY** are in bytes.

- The **DEFAULT** clause specifies a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as **NOW()** or **CURRENT_DATE**. The exception is that you can specify **CURRENT_TIMESTAMP** as the default for a **TIMESTAMP** column. See [Section 10.3.1.1, “TIMESTAMP Properties”](#).

If a column definition includes no explicit **DEFAULT** value, MySQL determines the default value as described in [Section 10.1.4, “Data Type Default Values”](#).

BLOB and **TEXT** columns cannot be assigned a default value.

- A comment for a column can be specified with the **COMMENT** option, up to 255 characters long. The comment is displayed by the **SHOW CREATE TABLE** and **SHOW FULL COLUMNS** statements.

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#).

Beginning with MySQL 5.1.19-ndb-6.2.5 and MySQL 5.1.20-ndb-6.3.2, it is also possible to specify a data storage format for indi-

vidual columns of **NDB** tables using **COLUMN_FORMAT**. Allowable column formats are **FIXED**, **DYNAMIC**, and **DEFAULT**. **FIXED** is used to specify fixed-width storage, **DYNAMIC** allows the column to be variable-width, and **DEFAULT** causes the column to use fixed-width or variable-width storage as determined by the column's data type (possibly overridden by a **ROW_FORMAT** specifier).

For **NDB** tables, the default value for **COLUMN_FORMAT** is **DEFAULT**.

COLUMN_FORMAT has no effect on columns of tables using storage engines other than **NDB**.

The following information applies to all MySQL Cluster users.

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

For **NDB** tables, beginning with MySQL 5.1.19-ndb-6.2.5 and MySQL 5.1.20-ndb-6.3.2, it is also possible to specify whether the column is stored on disk or in memory by using a **STORAGE** clause. **STORAGE DISK** causes the column to be stored on disk, and **STORAGE MEMORY** causes in-memory storage to be used.

For **NDB** tables, the default is **MEMORY**.

The **STORAGE** clause has no effect on tables using storage engines other than **NDB**.

The following information applies to all MySQL Cluster users.

- **KEY** is normally a synonym for **INDEX**. The key attribute **PRIMARY KEY** can also be specified as just **KEY** when given in a column definition. This was implemented for compatibility with other database systems.
 - A **UNIQUE** index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. For all engines, a **UNIQUE** index allows multiple **NULL** values for columns that can contain **NULL**.
 - A **PRIMARY KEY** is a unique index where all key columns must be defined as **NOT NULL**. If they are not explicitly declared as **NOT NULL**, MySQL declares them so implicitly (and silently). A table can have only one **PRIMARY KEY**. If you do not have a **PRIMARY KEY** and an application asks for the **PRIMARY KEY** in your tables, MySQL returns the first **UNIQUE** index that has no **NULL** columns as the **PRIMARY KEY**.
- In **InnoDB** tables, having a long **PRIMARY KEY** wastes a lot of space. (See Section 13.5.13, “InnoDB Table and Index Structures”.)
- In the created table, a **PRIMARY KEY** is placed first, followed by all **UNIQUE** indexes, and then the non-unique indexes. This helps the MySQL optimizer to prioritize which index to use and also more quickly to detect duplicated **UNIQUE** keys.
 - A **PRIMARY KEY** can be a multiple-column index. However, you cannot create a multiple-column index using the **PRIMARY KEY** key attribute in a column specification. Doing so only marks that single column as primary. You must use a separate **PRIMARY KEY(index_col_name, ...)** clause.
 - If a **PRIMARY KEY** or **UNIQUE** index consists of only one column that has an integer type, you can also refer to the column as **_rowid** in **SELECT** statements.
 - In MySQL, the name of a **PRIMARY KEY** is **PRIMARY**. For other indexes, if you do not assign a name, the index is assigned the same name as the first indexed column, with an optional suffix (**_2**, **_3**, ...) to make it unique. You can see index names for a table using **SHOW INDEX FROM tbl_name**. See Section 12.5.4.18, “SHOW INDEX Syntax”.
 - Some storage engines allow you to specify an index type when creating an index. The syntax for the *index_type* specifier is **USING type_name**.

Example:

```
CREATE TABLE lookup
(id INT, INDEX USING BTREE (id))
ENGINE = MEMORY;
```

Before MySQL 5.1.10, **USING** can be given only before the index column list. As of 5.1.10, the preferred position is after the column list. Use of the option before the column list will no longer be recognized as of MySQL 5.3.

index_option values specify additional options for an index. **USING** is one such option. For details about allowable *index_option* values, see [Section 12.1.7, “CREATE INDEX Syntax”](#).

For more information about indexes, see [Section 6.4.5, “How MySQL Uses Indexes”](#).

- In MySQL 5.1, only the **MyISAM**, **InnoDB**, and **MEMORY** storage engines support indexes on columns that can have **NULL** values. In other cases, you must declare indexed columns as **NOT NULL** or an error results.
- For **CHAR**, **VARCHAR**, **BINARY**, and **VARBINARY** columns, indexes can be created that use only the leading part of column values, using *col_name(length)* syntax to specify an index prefix length. **BLOB** and **TEXT** columns also can be indexed, but a prefix length *must* be given. Prefix lengths are given in characters for non-binary string types and in bytes for binary string types. That is, index entries consist of the first *length* characters of each column value for **CHAR**, **VARCHAR**, and **TEXT** columns, and the first *length* bytes of each column value for **BINARY**, **VARBINARY**, and **BLOB** columns. Indexing only a prefix of column values like this can make the index file much smaller. See [Section 6.4.3, “Column Indexes”](#).

Only the **MyISAM** and **InnoDB** storage engines support indexing on **BLOB** and **TEXT** columns. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for **InnoDB** tables). Note that prefix limits are measured in bytes, whereas the prefix length in **CREATE TABLE** statements is interpreted as number of characters for non-binary data types (**CHAR**, **VARCHAR**, **TEXT**). Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

- An *index_col_name* specification can end with **ASC** or **DESC**. These keywords are allowed for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.
- When you use **ORDER BY** or **GROUP BY** on a **TEXT** or **BLOB** column in a **SELECT**, the server sorts values using only the initial number of bytes indicated by the *max_sort_length* system variable. See [Section 10.4.3, “The BLOB and TEXT Types”](#).
- You can create special **FULLTEXT** indexes, which are used for full-text searches. Only the **MyISAM** storage engine supports **FULLTEXT** indexes. They can be created only from **CHAR**, **VARCHAR**, and **TEXT** columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See [Section 11.8, “Full-Text Search Functions”](#), for details of operation. A **WITH PARSER** clause can be specified as an *index_option* value to associate a parser plugin with the index if full-text indexing and searching operations need special handling. This clause is legal only for **FULLTEXT** indexes. See [Section 28.2, “The MySQL Plugin Interface”](#), for details on creating plugins.
- You can create **SPATIAL** indexes on spatial data types. Spatial types are supported only for **MyISAM** tables and indexed columns must be declared as **NOT NULL**. See [Chapter 18, Spatial Extensions](#).
- **InnoDB** tables support checking of foreign key constraints. See [Section 13.5, “The InnoDB Storage Engine”](#). Note that the **FOREIGN KEY** syntax in **InnoDB** is more restrictive than the syntax presented for the **CREATE TABLE** statement at the beginning of this section: The columns of the referenced table must always be explicitly named. **InnoDB** supports both **ON DELETE** and **ON UPDATE** actions on foreign keys. For the precise syntax, see [Section 13.5.6.4, “FOREIGN KEY Constraints”](#).

For other storage engines, MySQL Server parses and ignores the **FOREIGN KEY** and **REFERENCES** syntax in **CREATE TABLE** statements. The **CHECK** clause is parsed but ignored by all storage engines. See [Section 1.9.5.4, “Foreign Keys”](#).

Important

The inline **REFERENCES** specifications where the references are defined as part of the column specification are silently ignored by **InnoDB**. **InnoDB** only accepts **REFERENCES** clauses when specified as part of a separate **FOREIGN KEY** specification.

- For **MyISAM** tables, each **NULL** column takes one bit extra, rounded up to the nearest byte. The maximum row length in bytes can be calculated as follows:

```
row length = 1
              + (sum of column lengths)
              + (number of NULL columns + delete_flag + 7)/8
              + (number of variable-length columns)
```

delete_flag is 1 for tables with static row format. Static tables use a bit in the row record for a flag that indicates whether the row has been deleted. *delete_flag* is 0 for dynamic tables because the flag is stored in the dynamic row header.

These calculations do not apply for **InnoDB** tables, for which storage size is no different for **NULL** columns than for **NOT NULL** columns.

The **TABLESPACE ... STORAGE DISK** table option is used only with **NDB Cluster** tables. It assigns the table to a Cluster Disk Data tablespace. The tablespace named *tablespace_name* must already have been created using **CREATE TABLESPACE**. This table option was introduced in MySQL 5.1.6. See [Section 16.11, “MySQL Cluster Disk Data Tables”](#).

The **ENGINE** table option specifies the storage engine for the table.

The **ENGINE** table option takes the storage engine names shown in the following table.

Storage Engine	Description
ARCHIVE	The archiving storage engine. See Section 13.10, “The ARCHIVE Storage Engine” .
CSV	Tables that store rows in comma-separated values format. See Section 13.11, “The CSV Storage Engine” .
EXAMPLE	An example engine. See Section 13.8, “The EXAMPLE Storage Engine” .
FEDERATED	Storage engine that accesses remote tables. See Section 13.9, “The FEDERATED Storage Engine” .
HEAP	This is a synonym for MEMORY .
ISAM (OBSOLETE)	Not available in MySQL 5.1. If you are upgrading to MySQL 5.1 from a previous version, you should convert any existing ISAM tables to MyISAM before performing the upgrade.
InnoDB	Transaction-safe tables with row locking and foreign keys. See Section 13.5, “The InnoDB Storage Engine” .
MEMORY	The data for this storage engine is stored only in memory. See Section 13.7, “The MEMORY (HEAP) Storage Engine” .
MERGE	A collection of MyISAM tables used as one table. Also known as MRG_MyISAM . See Section 13.6, “The MERGE Storage Engine” .
MyISAM	The binary portable storage engine that is the default storage engine used by MySQL. See Section 13.4, “The MyISAM Storage Engine” .
NDBCLUSTER	Clustered, fault-tolerant, memory-based tables. Also known as NDB . See Chapter 16, MySQL Cluster .

If a storage engine is specified that is not available, MySQL uses the default engine instead. Normally, this is **MyISAM**. For example, if a table definition includes the **ENGINE=INNODB** option but the MySQL server does not support **INNODB** tables, the table is created as a **MyISAM** table. This makes it possible to have a replication setup where you have transactional tables on the master but tables created on the slave are non-transactional (to get more speed). In MySQL 5.1, a warning occurs if the storage engine specification is not honored.

Engine substitution can be controlled by the setting of the **NO_ENGINE_SUBSTITUTION** SQL mode, as described in [Section 5.2.6, “SQL Modes”](#).

Note

The older **TYPE** option was synonymous with **ENGINE**. **TYPE** has been deprecated since MySQL 4.0 but is still supported for backwards compatibility in MySQL 5.1 (excepting MySQL 5.1.7). Since MySQL 5.1.8, it produces a warning. It is scheduled for removal in MySQL 5.2. *You should not use **TYPE** in any new applications, and you should immediately begin conversion of existing applications to use **ENGINE** instead.* (See [Section C.1.22, “Changes in release 5.1.8 \(Not released\)”](#).)

The other table options are used to optimize the behavior of the table. In most cases, you do not have to specify any of them. These options apply to all storage engines unless otherwise indicated. Options that do not apply to a given storage engine may be accepted and remembered as part of the table definition. Such options then apply if you later use **ALTER TABLE** to convert the table to use a different storage engine.

- AUTO_INCREMENT**

The initial **AUTO_INCREMENT** value for the table. In MySQL 5.1, this works for **MyISAM**, **MEMORY**, and **InnoDB** tables. It also

works for [ARCHIVE](#) tables as of MySQL 5.1.6. To set the first auto-increment value for engines that do not support the [AUTO_INCREMENT](#) table option, insert a “dummy” row with a value one less than the desired value after creating the table, and then delete the dummy row.

For engines that support the [AUTO_INCREMENT](#) table option in [CREATE TABLE](#) statements, you can also use [ALTER TABLE tbl_name AUTO_INCREMENT = N](#) to reset the [AUTO_INCREMENT](#) value. The value cannot be set lower than the maximum value currently in the column.

- [AVG_ROW_LENGTH](#)

An approximation of the average row length for your table. You need to set this only for large tables with variable-size rows.

When you create a [MyISAM](#) table, MySQL uses the product of the [MAX_ROWS](#) and [AVG_ROW_LENGTH](#) options to decide how big the resulting table is. If you don't specify either option, the maximum size for a table is 256TB of data by default. (If your operating system does not support files that large, table sizes are constrained by the file size limit.) If you want to keep down the pointer sizes to make the index smaller and faster and you don't really need big files, you can decrease the default pointer size by setting the [myisam_data_pointer_size](#) system variable. (See [Section 5.2.3, “System Variables”](#).) If you want all your tables to be able to grow above the default limit and are willing to have your tables slightly slower and larger than necessary, you can increase the default pointer size by setting this variable. Setting the value to 7 allows table sizes up to 65,536TB.

- [\[DEFAULT\] CHARACTER SET](#)

Specify a default character set for the table. [CHARSET](#) is a synonym for [CHARACTER SET](#).

- [CHECKSUM](#)

Set this to 1 if you want MySQL to maintain a live checksum for all rows (that is, a checksum that MySQL updates automatically as the table changes). This makes the table a little slower to update, but also makes it easier to find corrupted tables. The [CHECKSUM TABLE](#) statement reports the checksum. ([MyISAM](#) only.)

- [COLLATE](#)

Specify a default collation for the table.

- [COMMENT](#)

A comment for the table, up to 60 characters long.

- [CONNECTION](#)

The connection string for a [FEDERATED](#) table.

Note

Older versions of MySQL used a [COMMENT](#) option for the connection string.

- [DATA DIRECTORY, INDEX DIRECTORY](#)

By using [DATA DIRECTORY='directory'](#) or [INDEX DIRECTORY='directory'](#) you can specify where the [MyISAM](#) storage engine should put a table's data file and index file. The directory must be the full pathname to the directory, not a relative path.

These options work only when you are not using the [--skip-symbolic-links](#) option. Your operating system must also have a working, thread-safe [realpath\(\)](#) call. See [Section 6.6.1.2, “Using Symbolic Links for Tables on Unix”](#), for more complete information.

If a [MyISAM](#) table is created with no [DATA DIRECTORY](#) option, the [.MYD](#) file is created in the database directory. By default, if [MyISAM](#) finds an existing [.MYD](#) file in this case, it overwrites it. The same applies to [.MYI](#) files for tables created with no [INDEX DIRECTORY](#) option. As of MySQL 5.1.23, to suppress this behavior, start the server with the [--keep_files_on_create](#) option, in which case [MyISAM](#) will not overwrite existing files and returns an error instead.

If a [MyISAM](#) table is created with a [DATA DIRECTORY](#) or [INDEX DIRECTORY](#) option and an existing [.MYD](#) or [.MYI](#) file is found, [MyISAM](#) always returns an error. It will not overwrite a file in the specified directory.

- [DELAY_KEY_WRITE](#)

Set this to 1 if you want to delay key updates for the table until the table is closed. See the description of the [delay_key_write](#) system variable in [Section 5.2.3, “System Variables”](#). (MyISAM only.)

- **INSERT_METHOD**

If you want to insert data into a [MERGE](#) table, you must specify with [INSERT_METHOD](#) the table into which the row should be inserted. [INSERT_METHOD](#) is an option useful for [MERGE](#) tables only. Use a value of [FIRST](#) or [LAST](#) to have inserts go to the first or last table, or a value of [NO](#) to prevent inserts. See [Section 13.6, “The MERGE Storage Engine”](#).

- **KEY_BLOCK_SIZE**

This option provides a hint to the storage engine about the size to use for index key blocks. The engine is allowed to change the value if necessary. A value of 0 indicates that the default value should be used. Individual index definitions can specify a [KEY_BLOCK_SIZE](#) value of their own to override the table value. [KEY_BLOCK_SIZE](#) was added in MySQL 5.1.10.

- **MAX_ROWS**

The maximum number of rows you plan to store in the table. This is not a hard limit, but rather a hint to the storage engine that the table must be able to store at least this many rows.

- **MIN_ROWS**

The minimum number of rows you plan to store in the table.

- **PACK_KEYS**

[PACK_KEYS](#) takes effect only with [MyISAM](#) tables. Set this option to 1 if you want to have smaller indexes. This usually makes updates slower and reads faster. Setting the option to 0 disables all packing of keys. Setting it to [DEFAULT](#) tells the storage engine to pack only long [CHAR](#) or [VARCHAR](#) columns.

If you do not use [PACK_KEYS](#), the default is to pack strings, but not numbers. If you use [PACK_KEYS=1](#), numbers are packed as well.

When packing binary number keys, MySQL uses prefix compression:

- Every key needs one extra byte to indicate how many bytes of the previous key are the same for the next key.
- The pointer to the row is stored in high-byte-first order directly after the key, to improve compression.

This means that if you have many equal keys on two consecutive rows, all following “same” keys usually only take two bytes (including the pointer to the row). Compare this to the ordinary case where the following keys takes [storage_size_for_key + pointer_size](#) (where the pointer size is usually 4). Conversely, you get a significant benefit from prefix compression only if you have many numbers that are the same. If all keys are totally different, you use one byte more per key, if the key is not a key that can have [NULL](#) values. (In this case, the packed key length is stored in the same byte that is used to mark if a key is [NULL](#).)

- **PASSWORD**

This option is unused. If you have a need to scramble your [.frm](#) files and make them unusable to any other MySQL server, please contact our sales department.

- **RAID_TYPE**

[RAID](#) support has been removed as of MySQL 5.0. For information on [RAID](#), see <http://dev.mysql.com/doc/refman/4.1/en/create-table.html>.

- **ROW_FORMAT**

Defines how the rows should be stored. For [MyISAM](#) tables, the option value can be [FIXED](#) or [DYNAMIC](#) for static or variable-length row format. [myisampack](#) sets the type to [COMPRESSED](#). See [Section 13.4.3, “MyISAM Table Storage Formats”](#).

For [InnoDB](#) tables, rows are stored in compact format ([ROW_FORMAT=COMPACT](#)) by default. The non-compact format used in older versions of MySQL can still be requested by specifying [ROW_FORMAT=REDUNDANT](#).

■ **Note**

During **CREATE TABLE**, if you specify a row format that the engine does support, the table will be created using the storage engines default row format. The information reported in this column in response to **SHOW TABLE STATUS** is the actual row format used. This may differ from the value in the **Create_options** column because the original **CREATE TABLE** definition is retained during creation.

- **UNION**

UNION is used when you want to access a collection of identical **MyISAM** tables as one. This works only with **MERGE** tables. See [Section 13.6, “The MERGE Storage Engine”](#).

You must have **SELECT**, **UPDATE**, and **DELETE** privileges for the tables you map to a **MERGE** table.

Note

Formerly, all tables used had to be in the same database as the **MERGE** table itself. This restriction no longer applies.)

partition_options can be used to control partitioning of the table created with **CREATE TABLE**.

Important

Not all options shown in the syntax for *partition_options* at the beginning of this section are available for all partitioning types. Please see the listings for the following individual types for information specific to each type, and see [Chapter 17, Partitioning](#), for more complete information about the workings of and uses for partitioning in MySQL, as well as additional examples of table creation and other statements relating to MySQL partitioning.

If used, *partition_options* is preceded by a **PARTITION BY** clause. This clause contains the function that is used to determine the partition; the function returns an integer value ranging from 1 to *num*, where *num* is the number of partitions. (The maximum number of user-defined partitions which a table may contain is 1024; the number of subpartitions — discussed later in this section — is included in this maximum.) The choices that are available for this function in MySQL 5.1 are shown in the following list:

- **HASH(*expr*)**: Hashes one or more columns to create a key for placing and locating rows. *expr* is an expression using one or more table columns. This can be any legal MySQL expression (including MySQL functions) that yields a single integer value. For example, these are all valid **CREATE TABLE** statements using **PARTITION BY HASH**:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5))
PARTITION BY HASH(col1);

CREATE TABLE t1 (col1 INT, col2 CHAR(5))
PARTITION BY HASH( ORD(col2) );

CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATETIME)
PARTITION BY HASH ( YEAR(col3) );
```

You may not use either **VALUES LESS THAN** or **VALUES IN** clauses with **PARTITION BY HASH**.

PARTITION BY HASH uses the remainder of *expr* divided by the number of partitions (that is, the modulus). For examples and additional information, see [Section 17.2.3, “HASH Partitioning”](#).

The **LINEAR** keyword entails a somewhat different algorithm. In this case, the number of the partition in which a row is stored is calculated as the result of one or more logical **AND** operations. For discussion and examples of linear hashing, see [Section 17.2.3.1, “LINEAR HASH Partitioning”](#).

- **KEY(*column_list*)**: This is similar to **HASH**, except that MySQL supplies the hashing function so as to guarantee an even data distribution. The *column_list* argument is simply a list of table columns. This example shows a simple table partitioned by key, with 4 partitions:

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY KEY(col3)
PARTITIONS 4;
```

For tables that are partitioned by key, you can employ linear partitioning by using the **LINEAR** keyword. This has the same effect as with tables that are partitioned by **HASH**. That is, the partition number is found using the **&** operator rather than the modulus (see [Section 17.2.3.1, “LINEAR HASH Partitioning”](#), and [Section 17.2.4, “KEY Partitioning”](#), for details). This example uses linear partitioning by key to distribute data between 5 partitions:

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY LINEAR KEY(col3)
PARTITIONS 5;
```

You may not use either **VALUES LESS THAN** or **VALUES IN** clauses with **PARTITION BY KEY**.

- **RANGE**: In this case, *expr* shows a range of values using a set of **VALUES LESS THAN** operators. When using range partitioning, you must define at least one partition using **VALUES LESS THAN**. You cannot use **VALUES IN** with range partitioning.

VALUES LESS THAN can be used with either a literal value or an expression that evaluates to a single value.

Suppose that you have a table that you wish to partition on a column containing year values, according to the following scheme:

Partition Number:	Years Range:
0	1990 and earlier
1	1991 – 1994
2	1995 – 1998
3	1999 – 2002
4	2003 – 2005
5	2006 and later

A table implementing such a partitioning scheme can be realized by the **CREATE TABLE** statement shown here:

```
CREATE TABLE t1 (
  year_col INT,
  some_data INT
)
PARTITION BY RANGE (year_col) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1995),
  PARTITION p2 VALUES LESS THAN (1999),
  PARTITION p3 VALUES LESS THAN (2002),
  PARTITION p4 VALUES LESS THAN (2006),
  PARTITION p5 VALUES LESS THAN MAXVALUE
);
```

PARTITION ... VALUES LESS THAN ... statements work in a consecutive fashion. **VALUES LESS THAN MAXVALUE** works to specify “leftover” values that are greater than the maximum value otherwise specified.

Note that **VALUES LESS THAN** clauses work sequentially in a manner similar to that of the **case** portions of a **switch ... case** block (as found in many programming languages such as C, Java, and PHP). That is, the clauses must be arranged in such a way that the upper limit specified in each successive **VALUES LESS THAN** is greater than that of the previous one, with the one referencing **MAXVALUE** coming last of all in the list.

- **LIST(*expr*)**: This is useful when assigning partitions based on a table column with a restricted set of possible values, such as a state or country code. In such a case, all rows pertaining to a certain state or country can be assigned to a single partition, or a partition can be reserved for a certain set of states or countries. It is similar to **RANGE**, except that only **VALUES IN** may be used to specify allowable values for each partition.

VALUES IN is used with a list of values to be matched. For instance, you could create a partitioning scheme such as the following:

```
CREATE TABLE client_firms (
  id INT,
  name VARCHAR(35)
)
PARTITION BY LIST (id) (
  PARTITION r0 VALUES IN (1, 5, 9, 13, 17, 21),
  PARTITION r1 VALUES IN (2, 6, 10, 14, 18, 22),
  PARTITION r2 VALUES IN (3, 7, 11, 15, 19, 23),
  PARTITION r3 VALUES IN (4, 8, 12, 16, 20, 24)
);
```

When using list partitioning, you must define at least one partition using **VALUES IN**. You cannot use **VALUES LESS THAN** with **PARTITION BY LIST**.

Note

Currently, the value list used with **VALUES IN** must consist of integer values only.

- The number of partitions may optionally be specified with a **PARTITIONS num** clause, where *num* is the number of partitions. If both this clause *and* any **PARTITION** clauses are used, *num* must be equal to the total number of any partitions that are declared using **PARTITION** clauses.

Note

Whether or not you use a **PARTITIONS** clause in creating a table that is partitioned by **RANGE** or **LIST**, you must still include at least one **PARTITION VALUES** clause in the table definition (see below).

- A partition may optionally be divided into a number of subpartitions. This can be indicated by using the optional **SUBPARTITION BY** clause. Subpartitioning may be done by **HASH** or **KEY**. Either of these may be **LINEAR**. These work in the same way as previously described for the equivalent partitioning types. (It is not possible to subpartition by **LIST** or **RANGE**.)

The number of subpartitions can be indicated using the **SUBPARTITIONS** keyword followed by an integer value.

- MySQL 5.1.12 introduces rigorous checking of the value used in a **PARTITIONS** or **SUBPARTITIONS** clause. Beginning with this version, this value must adhere to the following rules:
 - The value must be a positive, non-zero integer.
 - No leading zeroes are permitted.
 - The value must be an integer literal, and cannot not be an expression. For example, **PARTITIONS 0.2E+01** is not allowed, even though **0.2E+01** evaluates to **2**. (Bug#15890)

Each partition may be individually defined using a *partition_definition* clause. The individual parts making up this clause are as follows:

- PARTITION partition_name**: This specifies a logical name for the partition.
- A **VALUES** clause: For range partitioning, each partition must include a **VALUES LESS THAN** clause; for list partitioning, you must specify a **VALUES IN** clause for each partition. This is used to determine which rows are to be stored in this partition. See the discussions of partitioning types in [Chapter 17, Partitioning](#), for syntax examples.
- An optional **COMMENT** clause may be used to describe the partition. The comment must be set off in single quotes. Example:

```
COMMENT = 'Data for the years previous to 1999'
```

- DATA DIRECTORY** and **INDEX DIRECTORY** may be used to indicate the directory where, respectively, the data and indexes for this partition are to be stored. Both the *data_dir* and the *index_dir* must be absolute system pathnames. Example:

```
CREATE TABLE th (id INT, name VARCHAR(30), adate DATE)
PARTITION BY LIST(YEAR(adate))
(
  PARTITION p1999 VALUES IN (1995, 1999, 2003)
    DATA DIRECTORY = '/var/appdata/95/data'
    INDEX DIRECTORY = '/var/appdata/95/idx',
  PARTITION p2000 VALUES IN (1996, 2000, 2004)
    DATA DIRECTORY = '/var/appdata/96/data'
    INDEX DIRECTORY = '/var/appdata/96/idx',
  PARTITION p2001 VALUES IN (1997, 2001, 2005)
    DATA DIRECTORY = '/var/appdata/97/data'
    INDEX DIRECTORY = '/var/appdata/97/idx',
  PARTITION p2000 VALUES IN (1998, 2002, 2006)
    DATA DIRECTORY = '/var/appdata/98/data'
    INDEX DIRECTORY = '/var/appdata/98/idx'
);
```

DATA DIRECTORY and **INDEX DIRECTORY** behave in the same way as in the **CREATE TABLE** statement's *table_option* clause as used for **MyISAM** tables.

One data directory and one index directory may be specified per partition. If left unspecified, the data and indexes are stored by de-

fault in the MySQL data directory.

On Windows, paths used for `DATA DIRECTORY` and `INDEX DIRECTORY` must use the `/` (slash) character and not the `\` (backslash) character. See [Section 17.1, “Overview of Partitioning in MySQL”](#), for an example.

Note

Prior to MySQL 5.1.18, `DATA DIRECTORY` and `INDEX DIRECTORY` were allowed even if the `NO_DIR_IN_CREATE` server SQL mode was in effect at the time that a partitioned table was created. Beginning with MySQL 5.1.18, these options are ignored for creating partitioned tables if `NO_DIR_IN_CREATE` is in effect. ([Bug#24633](#))

- `MAX_ROWS` and `MIN_ROWS` may be used to specify, respectively, the maximum and minimum number of rows to be stored in the partition. The values for `max_number_of_rows` and `min_number_of_rows` must be positive integers. As with the table-level options with the same names, these act only as “suggestions” to the server and are not hard limits.
- The optional `TABLESPACE` clause may be used to designate a tablespace for the partition. Used for MySQL Cluster only.
- The partitioning handler accepts a `[STORAGE] ENGINE` option for both `PARTITION` and `SUBPARTITION`. Currently, the only way in which this can be used is to set all partitions or all subpartitions to the same storage engine, and an attempt to set different storage engines for partitions or subpartitions in the same table will give rise to the error `ERROR 1469 (HY000): THE MIX OF HANDLERS IN THE PARTITIONS IS NOT ALLOWED IN THIS VERSION OF MYSQL`. We expect to lift this restriction on partitioning in a future MySQL release.
- The `NODEGROUP` option can be used to make this partition act as part of the node group identified by `node_group_id`. This option is applicable only to MySQL Cluster.
- The partition definition may optionally contain one or more `subpartition_definition` clauses. Each of these consists at a minimum of the `SUBPARTITION name`, where `name` is an identifier for the subpartition. Except for the replacement of the `PARTITION` keyword with `SUBPARTITION`, the syntax for a subpartition definition is identical to that for a partition definition.

Subpartitioning must be done by `HASH` or `KEY`, and can be done only on `RANGE` or `LIST` partitions. See [Section 17.2.5, “Subpartitioning”](#).

Partitions can be modified, merged, added to tables, and dropped from tables. For basic information about the MySQL statements to accomplish these tasks, see [Section 12.1.4, “ALTER TABLE Syntax”](#). For more detailed descriptions and examples, see [Section 17.3, “Partition Management”](#).

Important

The original `CREATE TABLE` statement, including all specifications and table options are stored by MySQL when the table is created. The information is retained so that if you change storage engines, collations or other settings using an `ALTER TABLE` statement, the original table options specified are retained. This allows you to change between `InnoDB` and `MyISAM` table types even though the row formats supported by the two engines are different.

Because the text of the original statement is retained, but due to the way that certain values and options may be silently re-configured (such as the `ROW_FORMAT`), the active table definition (accessible through `DESCRIBE` or with `SHOW TABLE STATUS` and the table creation string (accessible through `SHOW CREATE TABLE`) will report different values.

You can create one table from another by adding a `SELECT` statement at the end of the `CREATE TABLE` statement:

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

MySQL creates new columns for all elements in the `SELECT`. For example:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
-> PRIMARY KEY (a), KEY(b))
-> ENGINE=MyISAM SELECT b,c FROM test2;
```

This creates a `MyISAM` table with three columns, `a`, `b`, and `c`. Notice that the columns from the `SELECT` statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
mysql> SELECT * FROM foo;
+---+
| n |
```

```

+---+
| 1 |
+---+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM bar;
+-----+-----+
| m      | n      |
+-----+-----+
| NULL   | 1      |
+-----+-----+
1 row in set (0.00 sec)

```

For each row in table `foo`, a row is inserted in `bar` with the values from `foo` and default values for the new columns.

In a table resulting from `CREATE TABLE ... SELECT`, columns named only in the `CREATE TABLE` part come first. Columns named in both parts or only in the `SELECT` part come after that. The data type of `SELECT` columns can be overridden by also specifying the column in the `CREATE TABLE` part.

If any errors occur while copying the data to the table, it is automatically dropped and not created.

`CREATE TABLE ... SELECT` does not automatically create any indexes for you. This is done intentionally to make the statement as flexible as possible. If you want to have indexes in the created table, you should specify these before the `SELECT` statement:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Some conversion of data types might occur. For example, the `AUTO_INCREMENT` attribute is not preserved, and `VARCHAR` columns can become `CHAR` columns.

When creating a table with `CREATE ... SELECT`, make sure to alias any function calls or expressions in the query. If you do not, the `CREATE` statement might fail or result in undesirable column names.

```

CREATE TABLE artists_and_works
SELECT artist.name, COUNT(work.artist_id) AS number_of_works
FROM artist LEFT JOIN work ON artist.id = work.artist_id
GROUP BY artist.id;

```

You can also explicitly specify the data type for a generated column:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

Use `LIKE` to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

The copy is created using the same version of the table storage format as the original table. The `SELECT` privilege is required on the original table.

`CREATE TABLE ... LIKE` does not preserve any `DATA DIRECTORY` or `INDEX DIRECTORY` table options that were specified for the original table, or any foreign key definitions.

You can precede the `SELECT` by `IGNORE` or `REPLACE` to indicate how to handle rows that duplicate unique key values. With `IGNORE`, new rows that duplicate an existing row on a unique key value are discarded. With `REPLACE`, new rows replace rows that have the same unique key value. If neither `IGNORE` nor `REPLACE` is specified, duplicate unique key values result in an error.

To ensure that the binary log can be used to re-create the original tables, MySQL does not allow concurrent inserts during `CREATE TABLE ... SELECT`.

12.1.10.1. Silent Column Specification Changes

In some cases, MySQL silently changes column specifications from those given in a `CREATE TABLE` or `ALTER TABLE` statement. These might be changes to a data type, to attributes associated with a data type, or to an index specification.

- `TIMESTAMP` display sizes are discarded.

Also note that **TIMESTAMP** columns are **NOT NULL** by default.

- Columns that are part of a **PRIMARY KEY** are made **NOT NULL** even if not declared that way.
- Trailing spaces are automatically deleted from **ENUM** and **SET** member values when the table is created.
- MySQL maps certain data types used by other SQL database vendors to MySQL types. See [Section 10.7, “Using Data Types from Other Database Engines”](#).
- If you include a **USING** clause to specify an index type that is not legal for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.

To see whether MySQL used a data type other than the one you specified, issue a **DESCRIBE** or **SHOW CREATE TABLE** statement after creating or altering the table.

Certain other data type changes can occur if you compress a table using **myisampack**. See [Section 13.4.3.3, “Compressed Table Characteristics”](#).

12.1.11. CREATE TABLESPACE Syntax

```
CREATE TABLESPACE tablespace
  ADD DATAFILE 'file'
  USE LOGFILE GROUP logfile_group
  [EXTENT_SIZE [=] extent_size]
  [INITIAL_SIZE [=] initial_size]
  [AUTOEXTEND_SIZE [=] autoextend_size]
  [MAX_SIZE [=] max_size]
  [NODEGROUP [=] nodegroup_id]
  [WAIT]
  [COMMENT [=] comment_text]
  ENGINE [=] engine
```

This statement is used to create a tablespace, which can contain one or more data files, providing storage space for tables. One data file is created and added to the tablespace using this statement. Additional data files may be added to the tablespace by using the **ALTER TABLESPACE** statement (see [Section 12.1.5, “ALTER TABLESPACE Syntax”](#)).

A log file group of one or more **UNDO** log files must be assigned to the tablespace to be created with the **USE LOGFILE GROUP** clause. *logfile_group* must be an existing log file group created with **CREATE LOGFILE GROUP** (see [Section 12.1.8, “CREATE LOGFILE GROUP Syntax”](#)). Multiple tablespaces may use the same log file group for **UNDO** logging.

The **EXTENT_SIZE** sets the size, in bytes, of the extents used by any files belonging to the tablespace. The default value is 1M. The minimum size is 32K, and the theoretical maximum is 2G, although the practical maximum size depends on a number of factors.

An *extent* is a unit of disk space allocation. One extent is filled with as much data as that extent can contain before another extent is used. In theory, up to 65,535 (64K) extents may be used per data file; however, the recommended maximum is 32,768 (32K). The recommended maximum size for a single data file is 32G — that is, 32K extents × 1 MB per extent. Smaller extents have the advantage that they tend to provide lower latency; however, larger extents tend to allow for greater throughput. You must also take into consideration that larger extents may mean longer node restart times. In addition, once an extent is allocated to a given table, it cannot be used to store data from another; an extent cannot store table data from more than one table. This means, for example, that a tablespace having a single datafile whose **INITIAL_SIZE** is 256 MB and whose **EXTENT_SIZE** is 128M has just two extents, and so can be used to store data from at most two different disk data tables.

You can see how many extents remain free in a given data file by querying the **INFORMATION_SCHEMA.FILES** table, and so derive an estimate for how much space remains free in the file. For further discussion and examples, see [Section 23.21, “The INFORMATION_SCHEMA FILES Table”](#).

The **INITIAL_SIZE** parameter sets the data file's total size in bytes. Once the file has been created, its size cannot be changed; however, you can add more data files to the tablespace using **ALTER TABLESPACE ... ADD DATAFILE**. See [Section 12.1.5, “ALTER TABLESPACE Syntax”](#).

INITIAL_SIZE is optional; its default value is **128M**.

When setting **EXTENT_SIZE** or **INITIAL_SIZE** (either or both), you may optionally follow the number with a one-letter abbreviation for an order of magnitude, similar to those used in **my.cnf**. Generally, this is one of the letters **M** (for megabytes) or **G** (for gigabytes).

`AUTOEXTEND_SIZE`, `MAX_SIZE`, `NODEGROUP`, `WAIT`, and `COMMENT` are parsed but ignored, and so have no effect in MySQL 5.1. These options are intended for future expansion.

The `ENGINE` parameter determines the storage engine which uses this tablespace, with `engine` being the name of the storage engine. In MySQL 5.1, `engine` must be one of the values `NDB` or `NDBCLUSTER`.

When `CREATE TABLESPACE` is used with `ENGINE = NDB`, a tablespace and associated data file are created on each Cluster data node. You can verify that the data files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_NAME, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE TABLESPACE_NAME = 'newts' AND FILE_TYPE = 'DATAFILE';
+-----+-----+-----+
| LOGFILE_GROUP_NAME | FILE_NAME | EXTRA |
+-----+-----+-----+
| lg_3               | newdata.dat | CLUSTER_NODE=3 |
| lg_3               | newdata.dat | CLUSTER_NODE=4 |
+-----+-----+-----+
2 rows in set (0.01 sec)
```

(See [Section 23.21](#), “The `INFORMATION_SCHEMA.FILES` Table”.)

`CREATE TABLESPACE` was added in MySQL 5.1.6. In MySQL 5.1, it is useful only with Disk Data storage for MySQL Cluster. See [Section 16.11](#), “MySQL Cluster Disk Data Tables”.

12.1.12. DROP DATABASE Syntax

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

`DROP DATABASE` drops all tables in the database and deletes the database. Be very careful with this statement! To use `DROP DATABASE`, you need the `DROP` privilege on the database. `DROP SCHEMA` is a synonym for `DROP DATABASE`.

Important

When a database is dropped, user privileges on the database are *not* automatically dropped. See [Section 12.5.1.3](#), “`GRANT Syntax`”.

`IF EXISTS` is used to prevent an error from occurring if the database does not exist.

If you use `DROP DATABASE` on a symbolically linked database, both the link and the original database are deleted.

`DROP DATABASE` returns the number of tables that were removed. This corresponds to the number of `.frm` files removed.

The `DROP DATABASE` statement removes from the given database directory those files and directories that MySQL itself may create during normal operation:

- All files with these extensions:

<code>.BAK</code>	<code>.DAT</code>	<code>.HSH</code>	<code>.MRG</code>
<code>.MYD</code>	<code>.MYI</code>	<code>.TRG</code>	<code>.TRN</code>
<code>.db</code>	<code>.frm</code>	<code>.ibd</code>	<code>.ndb</code>
<code>.par</code>			

- The `db.opt` file, if it exists.

If other files or directories remain in the database directory after MySQL removes those just listed, the database directory cannot be removed. In this case, you must remove any remaining files or directories manually and issue the `DROP DATABASE` statement again.

You can also drop databases with `mysqladmin`. See [Section 7.10](#), “`mysqladmin` — Client for Administering a MySQL Server”.

12.1.13. DROP INDEX Syntax

```
DROP INDEX index_name ON tbl_name
```

DROP INDEX drops the index named *index_name* from the table *tbl_name*. This statement is mapped to an **ALTER TABLE** statement to drop the index. See [Section 12.1.4, “ALTER TABLE Syntax”](#).

12.1.14. DROP LOGFILE GROUP Syntax

```
DROP LOGFILE GROUP logfile_group
ENGINE [=] engine
```

This statement drops the log file group named *logfile_group*. The log file group must already exist or an error results. (For information on creating log file groups, see [Section 12.1.8, “CREATE LOGFILE GROUP Syntax”](#).)

Important

Before dropping a log file group, you must drop all tablespaces that use that log file group for **UNDO** logging.

The required **ENGINE** clause provides the name of the storage engine used by the log file group to be dropped. In MySQL 5.1, the only permitted values for *engine* are **NDB** and **NDBCLUSTER**.

DROP LOGFILE GROUP was added in MySQL 5.1.6. In MySQL 5.1, it is useful only with Disk Data storage for MySQL Cluster. See [Section 16.11, “MySQL Cluster Disk Data Tables”](#).

12.1.15. DROP SERVER Syntax

```
DROP SERVER [ IF EXISTS ] server_name
```

Drops the server definition for the server named *server_name*. The corresponding row within the `mysql.servers` table will be deleted. This statement requires the **SUPER** privilege.

Dropping a server for a table does not affect any **FEDERATED** tables that used this connection information when they were created. See [Section 12.1.9, “CREATE SERVER Syntax”](#).

DROP SERVER does not cause an automatic commit.

12.1.16. DROP TABLE Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS]
tbl_name [, tbl_name] ...
[RESTRICT | CASCADE]
```

DROP TABLE removes one or more tables. You must have the **DROP** privilege for each table. All table data and the table definition are *removed, so be careful* with this statement! If any of the tables named in the argument list do not exist, MySQL returns an error indicating by name which non-existing tables it was unable to drop, but it also drops all of the tables in the list that do exist.

Important

When a table is dropped, user privileges on the table are *not* automatically dropped. See [Section 12.5.1.3, “GRANT Syntax”](#).

Note that for a partitioned table, **DROP TABLE** permanently removes the table definition, all of its partitions, and all of the data which was stored in those partitions. It also removes the partitioning definition (`.par`) file associated with the dropped table.

Use **IF EXISTS** to prevent an error from occurring for tables that do not exist. A **NOTE** is generated for each non-existent table when using **IF EXISTS**. See [Section 12.5.4.32, “SHOW WARNINGS Syntax”](#).

RESTRICT and **CASCADE** are allowed to make porting easier. In MySQL 5.1, they do nothing.

Note

DROP TABLE automatically commits the current active transaction, unless you use the **TEMPORARY** keyword.

The **TEMPORARY** keyword has the following effects:

- The statement drops only **TEMPORARY** tables.
- The statement does not end an ongoing transaction.
- No access rights are checked. (A **TEMPORARY** table is visible only to the client that created it, so no check is necessary.)

Using **TEMPORARY** is a good way to ensure that you do not accidentally drop a non-**TEMPORARY** table.

12.1.17. DROP TABLESPACE Syntax

```
DROP TABLESPACE tablespace
ENGINE [=] engine
```

This statement drops a tablespace that was previously created using **CREATE TABLESPACE** (see [Section 12.1.11, “CREATE TABLESPACE Syntax”](#)).

Important

The tablespace to be dropped must not contain any data files; in other words, before you can drop a tablespace, you must first drop each of its data files using **ALTER TABLESPACE ... DROP DATAFILE** (see [Section 12.1.5, “ALTER TABLESPACE Syntax”](#)).

The **ENGINE** clause (required) specifies the storage engine used by the tablespace. In MySQL 5.1, the only accepted values for *engine* are **NDB** and **NDBCLUSTER**.

DROP TABLESPACE was added in MySQL 5.1.6. In MySQL 5.1, it is useful only with Disk Data storage for MySQL Cluster. See [Section 16.11, “MySQL Cluster Disk Data Tables”](#).

12.1.18. RENAME DATABASE Syntax

```
RENAME {DATABASE | SCHEMA} db_name TO new_db_name;
```

This statement renames a database. It requires the **ALTER** and **DROP** privileges for the database, and the **CREATE** privilege for the new database. **RENAME SCHEMA** is a synonym for **RENAME DATABASE**.

When the server receives this statement, it creates a new database. Then it moves tables and other database objects such as triggers to the new database. It also updates the **Db** column in the system tables for objects such as stored routines and events. Finally, the server drops the old database.

Note that currently there are these limitations:

- **RENAME DATABASE** does not change any account privileges listed in the system tables. They must be changed manually.
- **RENAME DATABASE** destroys any stored routines or events associated with the original database.

This statement was added in MySQL 5.1.7.

12.1.19. RENAME TABLE Syntax

```
RENAME TABLE tbl_name TO new_tbl_name
[, tbl_name2 TO new_tbl_name2] ...
```

This statement renames one or more tables.

The rename operation is done atomically, which means that no other thread can access any of the tables while the rename is running. For example, if you have an existing table **old_table**, you can create another table **new_table** that has the same structure but is empty, and then replace the existing table with the empty one as follows (assuming that **backup_table** does not already exist):

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

If the statement renames more than one table, renaming operations are done from left to right. If you want to swap two table names, you can do so like this (assuming that `tmp_table` does not already exist):

```
RENAME TABLE old_table TO tmp_table,  
              new_table TO old_table,  
              tmp_table TO new_table;
```

As long as two databases are on the same filesystem, you can use `RENAME TABLE` to move a table from one database to another:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

If there are any triggers associated with a table which is moved to a different database using `RENAME TABLE`, then the statement fails with the error `TRIGGER IN WRONG SCHEMA`.

`RENAME TABLE` also works for views, as long as you do not try to rename a view into a different database.

Any privileges granted specifically for the renamed table or view are not migrated to the new name. They must be changed manually.

When you execute `RENAME`, you cannot have any locked tables or active transactions. You must also have the `ALTER` and `DROP` privileges on the original table, and the `CREATE` and `INSERT` privileges on the new table.

If MySQL encounters any errors in a multiple-table rename, it does a reverse rename for all renamed tables to return everything to its original state.

12.2. Data Manipulation Statements

12.2.1. DELETE Syntax

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name  
      [WHERE where_condition]  
      [ORDER BY ...]  
      [LIMIT row_count]
```

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]  
      tbl_name[.*] [, tbl_name[.*]] ...  
      FROM table_references  
      [WHERE where_condition]
```

Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]  
      FROM tbl_name[.*] [, tbl_name[.*]] ...  
      USING table_references  
      [WHERE where_condition]
```

For the single-table syntax, the `DELETE` statement deletes rows from `tbl_name` and returns the number of rows deleted. The `WHERE` clause, if given, specifies the conditions that identify which rows to delete. With no `WHERE` clause, all rows are deleted. If the `ORDER BY` clause is specified, the rows are deleted in the order that is specified. The `LIMIT` clause places a limit on the number of rows that can be deleted.

For the multiple-table syntax, `DELETE` deletes from each `tbl_name` the rows that satisfy the conditions. In this case, `ORDER BY` and `LIMIT` cannot be used.

`where_condition` is an expression that evaluates to true for each row to be deleted. It is specified as described in [Section 12.2.7, “SELECT Syntax”](#).

As stated, a `DELETE` statement with no `WHERE` clause deletes all rows. A faster way to do this, when you do not want to know the number of deleted rows, is to use `TRUNCATE TABLE`. See [Section 12.2.9, “TRUNCATE Syntax”](#).

If you delete the row containing the maximum value for an `AUTO_INCREMENT` column, the value is not reused for a `MyISAM` or `InnoDB` table. If you delete all rows in the table with `DELETE FROM tbl_name` (without a `WHERE` clause) in `AUTOCOMMIT` mode,

the sequence starts over for all storage engines except **InnoDB** and **MyISAM**. There are some exceptions to this behavior for **InnoDB** tables, as discussed in [Section 13.5.6.3, “How AUTO_INCREMENT Handling Works in InnoDB”](#).

For **MyISAM** tables, you can specify an **AUTO_INCREMENT** secondary column in a multiple-column key. In this case, reuse of values deleted from the top of the sequence occurs even for **MyISAM** tables. See [Section 3.6.9, “Using AUTO_INCREMENT”](#).

The **DELETE** statement supports the following modifiers:

- If you specify **LOW_PRIORITY**, the server delays execution of the **DELETE** until no other clients are reading from the table. This affects only storage engines that use only table-level locking (**MyISAM**, **MEMORY**, **MERGE**).
- For **MyISAM** tables, if you use the **QUICK** keyword, the storage engine does not merge index leaves during delete, which may speed up some kinds of delete operations.
- The **IGNORE** keyword causes MySQL to ignore all errors during the process of deleting rows. (Errors encountered during the parsing stage are processed in the usual manner.) Errors that are ignored due to the use of **IGNORE** are returned as warnings.

The speed of delete operations may also be affected by factors discussed in [Section 6.2.19, “Speed of DELETE Statements”](#).

In **MyISAM** tables, deleted rows are maintained in a linked list and subsequent **INSERT** operations reuse old row positions. To reclaim unused space and reduce file sizes, use the **OPTIMIZE TABLE** statement or the **myisamchk** utility to reorganize tables. **OPTIMIZE TABLE** is easier, but **myisamchk** is faster. See [Section 12.5.2.5, “OPTIMIZE TABLE Syntax”](#), and [Section 7.5, “myisamchk — MyISAM Table-Maintenance Utility”](#).

The **QUICK** modifier affects whether index leaves are merged for delete operations. **DELETE QUICK** is most useful for applications where index values for deleted rows are replaced by similar index values from rows inserted later. In this case, the holes left by deleted values are reused.

DELETE QUICK is not useful when deleted values lead to underfilled index blocks spanning a range of index values for which new inserts occur again. In this case, use of **QUICK** can lead to wasted space in the index that remains unreclaimed. Here is an example of such a scenario:

1. Create a table that contains an indexed **AUTO_INCREMENT** column.
2. Insert many rows into the table. Each insert results in an index value that is added to the high end of the index.
3. Delete a block of rows at the low end of the column range using **DELETE QUICK**.

In this scenario, the index blocks associated with the deleted index values become underfilled but are not merged with other index blocks due to the use of **QUICK**. They remain underfilled when new inserts occur, because new rows do not have index values in the deleted range. Furthermore, they remain underfilled even if you later use **DELETE** without **QUICK**, unless some of the deleted index values happen to lie in index blocks within or adjacent to the underfilled blocks. To reclaim unused index space under these circumstances, use **OPTIMIZE TABLE**.

If you are going to delete many rows from a table, it might be faster to use **DELETE QUICK** followed by **OPTIMIZE TABLE**. This rebuilds the index rather than performing many index block merge operations.

The MySQL-specific **LIMIT row_count** option to **DELETE** tells the server the maximum number of rows to be deleted before control is returned to the client. This can be used to ensure that a given **DELETE** statement does not take too much time. You can simply repeat the **DELETE** statement until the number of affected rows is less than the **LIMIT** value.

If the **DELETE** statement includes an **ORDER BY** clause, the rows are deleted in the order specified by the clause. This is really useful only in conjunction with **LIMIT**. For example, the following statement finds rows matching the **WHERE** clause, sorts them by **timestamp_column**, and deletes the first (oldest) one:

```
DELETE FROM somelog WHERE user = 'jcole'  
ORDER BY timestamp_column LIMIT 1;
```

You can specify multiple tables in a **DELETE** statement to delete rows from one or more tables depending on the particular condition in the **WHERE** clause. However, you cannot use **ORDER BY** or **LIMIT** in a multiple-table **DELETE**. The *table_references* clause lists the tables involved in the join. Its syntax is described in [Section 12.2.7.1, “JOIN Syntax”](#).

For the first multiple-table syntax, only matching rows from the tables listed before the **FROM** clause are deleted. For the second mul-

tuple-table syntax, only matching rows from the tables listed in the **FROM** clause (before the **USING** clause) are deleted. The effect is that you can delete rows from many tables at the same time and have additional tables that are used only for searching:

```
DELETE t1, t2 FROM t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

Or:

```
DELETE FROM t1, t2 USING t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

These statements use all three tables when searching for rows to delete, but delete matching rows only from tables **t1** and **t2**.

The preceding examples show inner joins that use the comma operator, but multiple-table **DELETE** statements can use any type of join allowed in **SELECT** statements, such as **LEFT JOIN**.

The syntax allows **. *** after the table names for compatibility with **Access**.

If you use a multiple-table **DELETE** statement involving **InnoDB** tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, you should delete from a single table and rely on the **ON DELETE** capabilities that **InnoDB** provides to cause the other tables to be modified accordingly.

Note

If you provide an alias for a table, you must use the alias when referring to the table:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

Cross-database deletes are supported for multiple-table deletes, but in this case, you must refer to the tables without using aliases. For example:

```
DELETE test1.tmp1, test2.tmp2 FROM test1.tmp1, test2.tmp2 WHERE ...
```

Currently, you cannot delete from a table and select from the same table in a subquery.

12.2.2. DO Syntax

```
DO expr [, expr] ...
```

DO executes the expressions but does not return any results. In most respects, **DO** is shorthand for **SELECT *expr*, ...**, but has the advantage that it is slightly faster when you do not care about the result.

DO is useful primarily with functions that have side effects, such as **RELEASE_LOCK()**.

12.2.3. HANDLER Syntax

```
HANDLER tbl_name OPEN [ [AS] alias ]
HANDLER tbl_name READ index_name { = | >= | <= | < } (value1,value2,...)
[ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
[ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
[ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name CLOSE
```

The **HANDLER** statement provides direct access to table storage engine interfaces. It is available for **MyISAM** and **InnoDB** tables.

The **HANDLER ... OPEN** statement opens a table, making it accessible via subsequent **HANDLER ... READ** statements. This table object is not shared by other threads and is not closed until the thread calls **HANDLER ... CLOSE** or the thread terminates. If you open the table using an alias, further references to the open table with other **HANDLER** statements must use the alias rather than the table name.

The first **HANDLER ... READ** syntax fetches a row where the index specified satisfies the given values and the **WHERE** condition is met. If you have a multiple-column index, specify the index column values as a comma-separated list. Either specify values for all the columns in the index, or specify values for a leftmost prefix of the index columns. Suppose that an index **my_idx** includes three columns named **col_a**, **col_b**, and **col_c**, in that order. The **HANDLER** statement can specify values for all three columns in the in-

dex, or for the columns in a leftmost prefix. For example:

```
HANDLER ... READ my_idx = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... READ my_idx = (col_a_val,col_b_val) ...
HANDLER ... READ my_idx = (col_a_val) ...
```

To employ the **HANDLER** interface to refer to a table's **PRIMARY KEY**, use the quoted identifier ``PRIMARY``:

```
HANDLER tbl_name READ `PRIMARY` ...
```

The second **HANDLER ... READ** syntax fetches a row from the table in index order that matches the **WHERE** condition.

The third **HANDLER ... READ** syntax fetches a row from the table in natural row order that matches the **WHERE** condition. It is faster than **HANDLER tbl_name READ index_name** when a full table scan is desired. Natural row order is the order in which rows are stored in a **MyISAM** table data file. This statement works for **InnoDB** tables as well, but there is no such concept because there is no separate data file.

Without a **LIMIT** clause, all forms of **HANDLER ... READ** fetch a single row if one is available. To return a specific number of rows, include a **LIMIT** clause. It has the same syntax as for the **SELECT** statement. See [Section 12.2.7, “SELECT Syntax”](#).

HANDLER ... CLOSE closes a table that was opened with **HANDLER ... OPEN**.

HANDLER is a somewhat low-level statement. For example, it does not provide consistency. That is, **HANDLER ... OPEN** does *not* take a snapshot of the table, and does *not* lock the table. This means that after a **HANDLER ... OPEN** statement is issued, table data can be modified (by the current thread or other threads) and these modifications might be only partially visible to **HANDLER ... NEXT** or **HANDLER ... PREV** scans.

There are several reasons to use the **HANDLER** interface instead of normal **SELECT** statements:

- **HANDLER** is faster than **SELECT**:
 - A designated storage engine handler object is allocated for the **HANDLER ... OPEN**. The object is reused for subsequent **HANDLER** statements for that table; it need not be reinitialized for each one.
 - There is less parsing involved.
 - There is no optimizer or query-checking overhead.
 - The table does not have to be locked between two handler requests.
 - The handler interface does not have to provide a consistent look of the data (for example, dirty reads are allowed), so the storage engine can use optimizations that **SELECT** does not normally allow.
- For applications that use a low-level **ISAM**-like interface, **HANDLER** makes it much easier to port them to MySQL.
- **HANDLER** enables you to traverse a database in a manner that is difficult (or even impossible) to accomplish with **SELECT**. The **HANDLER** interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database.

12.2.4. INSERT Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
VALUES ({expr | DEFAULT},...), (...),...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
SET col_name={expr | DEFAULT}, ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Or:


```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
SELECT ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

INSERT inserts new rows into an existing table. The **INSERT ... VALUES** and **INSERT ... SET** forms of the statement insert rows based on explicitly specified values. The **INSERT ... SELECT** form inserts rows selected from another table or tables. **INSERT ... SELECT** is discussed further in [Section 12.2.4.1, “INSERT ... SELECT Syntax”](#).

You can use **REPLACE** instead of **INSERT** to overwrite old rows. **REPLACE** is the counterpart to **INSERT IGNORE** in the treatment of new rows that contain unique key values that duplicate old rows: The new rows are used to replace the old rows rather than being discarded. See [Section 12.2.6, “REPLACE Syntax”](#).

tbl_name is the table into which rows should be inserted. The columns for which the statement provides values can be specified as follows:

- You can provide a comma-separated list of column names following the table name. In this case, a value for each named column must be provided by the **VALUES** list or the **SELECT** statement.
- If you do not specify a list of column names for **INSERT ... VALUES** or **INSERT ... SELECT**, values for every column in the table must be provided by the **VALUES** list or the **SELECT** statement. If you do not know the order of the columns in the table, use **DESCRIBE *tbl_name*** to find out.
- The **SET** clause indicates the column names explicitly.

Column values can be given in several ways:

- If you are not running in strict SQL mode, any column not explicitly given a value is set to its default (explicit or implicit) value. For example, if you specify a column list that does not name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in [Section 10.1.4, “Data Type Default Values”](#). See also [Section 1.9.6.2, “Constraints on Invalid Data”](#).

If you want an **INSERT** statement to generate an error unless you explicitly specify values for all columns that do not have a default value, you should use strict mode. See [Section 5.2.6, “SQL Modes”](#).

- Use the keyword **DEFAULT** to set a column explicitly to its default value. This makes it easier to write **INSERT** statements that assign values to all but a few columns, because it enables you to avoid writing an incomplete **VALUES** list that does not include a value for each column in the table. Otherwise, you would have to write out the list of column names corresponding to each value in the **VALUES** list.

You can also use **DEFAULT(*col_name*)** as a more general form that can be used in expressions to produce a given column's default value.

- If both the column list and the **VALUES** list are empty, **INSERT** creates a row with each column set to its default value:

```
INSERT INTO tbl_name () VALUES();
```

In strict mode, an error occurs if any column doesn't have a default value. Otherwise, MySQL uses the implicit default value for any column that does not have an explicitly defined default.

- You can specify an expression *expr* to provide a column value. This might involve type conversion if the type of the expression does not match the type of the column, and conversion of a given value can result in different inserted values depending on the data type. For example, inserting the string `'1999.0e-2'` into an **INT**, **FLOAT**, **DECIMAL(10,6)**, or **YEAR** column results in the values `1999`, `19.9921`, `19.992100`, and `1999` being inserted, respectively. The reason the value stored in the **INT** and **YEAR** columns is `1999` is that the string-to-integer conversion looks only at as much of the initial part of the string as may be considered a valid integer or year. For the floating-point and fixed-point columns, the string-to-floating-point conversion considers the entire string a valid floating-point value.

An expression *expr* can refer to any column that was set earlier in a value list. For example, you can do this because the value for `col2` refers to `col1`, which has previously been assigned:

```
INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

But the following is not legal, because the value for `col1` refers to `col2`, which is assigned after `col1`:

```
INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

One exception involves columns that contain `AUTO_INCREMENT` values. Because the `AUTO_INCREMENT` value is generated after other value assignments, any reference to an `AUTO_INCREMENT` column in the assignment returns a `0`.

`INSERT` statements that use `VALUES` syntax can insert multiple rows. To do this, include multiple lists of column values, each enclosed within parentheses and separated by commas. Example:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

The values list for each row must be enclosed within parentheses. The following statement is illegal because the number of values in the list does not match the number of column names:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

The rows-affected value for an `INSERT` can be obtained using the `mysql_affected_rows()` C API function. See [Section 25.2.3.1](#), “`mysql_affected_rows()`”.

If you use an `INSERT ... VALUES` statement with multiple value lists or `INSERT ... SELECT`, the statement returns an information string in this format:

```
Records: 100 Duplicates: 0 Warnings: 0
```

Records indicates the number of rows processed by the statement. (This is not necessarily the number of rows actually inserted because **Duplicates** can be non-zero.) **Duplicates** indicates the number of rows that could not be inserted because they would duplicate some existing unique index value. **Warnings** indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting `NULL` into a column that has been declared `NOT NULL`. For multiple-row `INSERT` statements or `INSERT INTO ... SELECT` statements, the column is set to the implicit default value for the column data type. This is `0` for numeric types, the empty string (`' '`) for string types, and the “zero” value for date and time types. `INSERT INTO ... SELECT` statements are handled the same way as multiple-row inserts because the server does not examine the result set from the `SELECT` to see whether it returns a single row. (For a single-row `INSERT`, no warning occurs when `NULL` is inserted into a `NOT NULL` column. Instead, the statement fails with an error.)
- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the closest endpoint of the range.
- Assigning a value such as `'10.34 a'` to a numeric column. The trailing non-numeric text is stripped off and the remaining numeric part is inserted. If the string value has no leading numeric part, the column is set to `0`.
- Inserting a string into a string column (`CHAR`, `VARCHAR`, `TEXT`, or `BLOB`) that exceeds the column's maximum length. The value is truncated to the column's maximum length.
- Inserting a value into a date or time column that is illegal for the data type. The column is set to the appropriate zero value for the type.

If you are using the C API, the information string can be obtained by invoking the `mysql_info()` function. See [Section 25.2.3.35](#), “`mysql_info()`”.

If `INSERT` inserts a row into a table that has an `AUTO_INCREMENT` column, you can find the value used for that column by using the SQL `LAST_INSERT_ID()` function. From within the C API, use the `mysql_insert_id()` function. However, you should note that the two functions do not always behave identically. The behavior of `INSERT` statements with respect to `AUTO_INCREMENT` columns is discussed further in [Section 11.11.3](#), “[Information Functions](#)”, and [Section 25.2.3.37](#), “`mysql_insert_id()`”.

The `INSERT` statement supports the following modifiers:

- If you use the `DELAYED` keyword, the server puts the row or rows to be inserted into a buffer, and the client issuing the `INSERT`

DELAYED statement can then continue immediately. If the table is in use, the server holds the rows. When the table is free, the server begins inserting rows, checking periodically to see whether there are any new read requests for the table. If there are, the delayed row queue is suspended until the table becomes free again. See [Section 12.2.4.2, “INSERT DELAYED Syntax”](#).

DELAYED is ignored with **INSERT ... SELECT** or **INSERT ... ON DUPLICATE KEY UPDATE**.

- If you use the **LOW_PRIORITY** keyword, execution of the **INSERT** is delayed until no other clients are reading from the table. This includes other clients that began reading while existing clients are reading, and while the **INSERT LOW_PRIORITY** statement is waiting. It is possible, therefore, for a client that issues an **INSERT LOW_PRIORITY** statement to wait for a very long time (or even forever) in a read-heavy environment. (This is in contrast to **INSERT DELAYED**, which lets the client continue at once. Note that **LOW_PRIORITY** should normally not be used with **MyISAM** tables because doing so disables concurrent inserts. See [Section 6.3.3, “Concurrent Inserts”](#)).

If you specify **HIGH_PRIORITY**, it overrides the effect of the `--low-priority-updates` option if the server was started with that option. It also causes concurrent inserts not to be used. See [Section 6.3.3, “Concurrent Inserts”](#).

LOW_PRIORITY and **HIGH_PRIORITY** affect only storage engines that use only table-level locking (**MyISAM**, **MEMORY**, **MERGE**).

- If you use the **IGNORE** keyword, errors that occur while executing the **INSERT** statement are treated as warnings instead. For example, without **IGNORE**, a row that duplicates an existing **UNIQUE** index or **PRIMARY KEY** value in the table causes a duplicate-key error and the statement is aborted. With **IGNORE**, the row still is not inserted, but no error is issued. Data conversions that would trigger errors abort the statement if **IGNORE** is not specified. With **IGNORE**, invalid values are adjusted to the closest values and inserted; warnings are produced but the statement does not abort. You can determine with the `mysql_info()` C API function how many rows were actually inserted into the table.
- If you specify **ON DUPLICATE KEY UPDATE**, and a row is inserted that would cause a duplicate value in a **UNIQUE** index or **PRIMARY KEY**, an **UPDATE** of the old row is performed. See [Section 12.2.4.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”](#).

12.2.4.1. INSERT ... SELECT Syntax

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
SELECT ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

With **INSERT ... SELECT**, you can quickly insert many rows into a table from one or many tables. For example:

```
INSERT INTO tbl_temp2 (fld_id)
SELECT tbl_temp1.fld_order_id
FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

The following conditions hold for a **INSERT ... SELECT** statements:

- Specify **IGNORE** to ignore rows that would cause duplicate-key violations.
- **DELAYED** is ignored with **INSERT ... SELECT**.
- The target table of the **INSERT** statement may appear in the **FROM** clause of the **SELECT** part of the query. (This was not possible in some older versions of MySQL.) In this case, MySQL creates a temporary table to hold the rows from the **SELECT** and then inserts those rows into the target table. However, it remains true that you cannot use **INSERT INTO t ... SELECT ... FROM t** when **t** is a **TEMPORARY** table, because **TEMPORARY** tables cannot be referred to twice in the same statement (see [Section B.1.7.3, “TEMPORARY TABLE Problems”](#)).
- **AUTO_INCREMENT** columns work as usual.
- To ensure that the binary log can be used to re-create the original tables, MySQL does not allow concurrent inserts for **INSERT ... SELECT** statements.
- Currently, you cannot insert into a table and select from the same table in a subquery.
- To avoid ambiguous column reference problems when the **SELECT** and the **INSERT** refer to the same table, provide a unique alias for each table used in the **SELECT** part, and qualify column names in that part with the appropriate alias.

In the values part of **ON DUPLICATE KEY UPDATE**, you can refer to columns in other tables, as long as you do not use **GROUP BY** in the **SELECT** part. One side effect is that you must qualify non-unique column names in the values part.

12.2.4.2. **INSERT DELAYED** Syntax

```
INSERT DELAYED ...
```

The **DELAYED** option for the **INSERT** statement is a MySQL extension to standard SQL that is very useful if you have clients that cannot or need not wait for the **INSERT** to complete. This is a common situation when you use MySQL for logging and you also periodically run **SELECT** and **UPDATE** statements that take a long time to complete.

When a client uses **INSERT DELAYED**, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

Another major benefit of using **INSERT DELAYED** is that inserts from many clients are bundled together and written in one block. This is much faster than performing many separate inserts.

Note that **INSERT DELAYED** is slower than a normal **INSERT** if the table is not otherwise in use. There is also the additional overhead for the server to handle a separate thread for each table for which there are delayed rows. This means that you should use **INSERT DELAYED** only when you are really sure that you need it.

The queued rows are held only in memory until they are inserted into the table. This means that if you terminate **mysqld** forcibly (for example, with **kill -9**) or if **mysqld** dies unexpectedly, *any queued rows that have not been written to disk are lost*.

There are some constraints on the use of **DELAYED**:

- **INSERT DELAYED** works only with **MyISAM**, **MEMORY**, **ARCHIVE**, and (as of MySQL 5.1.19) **BLACKHOLE** tables. See [Section 13.4, “The MyISAM Storage Engine”](#), [Section 13.7, “The MEMORY \(HEAP\) Storage Engine”](#), [Section 13.10, “The ARCHIVE Storage Engine”](#), and [Section 13.12, “The BLACKHOLE Storage Engine”](#).
- For **MyISAM** tables, if there are no free blocks in the middle of the data file, concurrent **SELECT** and **INSERT** statements are supported. Under these circumstances, you very seldom need to use **INSERT DELAYED** with **MyISAM**.
- **INSERT DELAYED** should be used only for **INSERT** statements that specify value lists. The server ignores **DELAYED** for **INSERT ... SELECT** or **INSERT ... ON DUPLICATE KEY UPDATE** statements.
- Because the **INSERT DELAYED** statement returns immediately, before the rows are inserted, you cannot use **LAST_INSERT_ID()** to get the **AUTO_INCREMENT** value that the statement might generate.
- **DELAYED** rows are not visible to **SELECT** statements until they actually have been inserted.
- **DELAYED** is ignored on slave replication servers because it could cause the slave to have different data than the master.
- Pending **INSERT DELAYED** statements are lost if a table is write locked and **ALTER TABLE** is used to modify the table structure.
- **INSERT DELAYED** is not supported for views.

The following describes in detail what happens when you use the **DELAYED** option to **INSERT** or **REPLACE**. In this description, the “thread” is the thread that received an **INSERT DELAYED** statement and “handler” is the thread that handles all **INSERT DELAYED** statements for a particular table.

- When a thread executes a **DELAYED** statement for a table, a handler thread is created to process all **DELAYED** statements for the table, if no such handler already exists.
- The thread checks whether the handler has previously acquired a **DELAYED** lock; if not, it tells the handler thread to do so. The **DELAYED** lock can be obtained even if other threads have a **READ** or **WRITE** lock on the table. However, the handler waits for all **ALTER TABLE** locks or **FLUSH TABLES** statements to finish, to ensure that the table structure is up to date.
- The thread executes the **INSERT** statement, but instead of writing the row to the table, it puts a copy of the final row into a queue that is managed by the handler thread. Any syntax errors are noticed by the thread and reported to the client program.
- The client cannot obtain from the server the number of duplicate rows or the **AUTO_INCREMENT** value for the resulting row, be-

cause the `INSERT` returns before the insert operation has been completed. (If you use the C API, the `mysql_info()` function does not return anything meaningful, for the same reason.)

- The binary log is updated by the handler thread when the row is inserted into the table. In case of multiple-row inserts, the binary log is updated when the first row is inserted.
- Each time that `delayed_insert_limit` rows are written, the handler checks whether any `SELECT` statements are still pending. If so, it allows these to execute before continuing.
- When the handler has no more rows in its queue, the table is unlocked. If no new `INSERT DELAYED` statements are received within `delayed_insert_timeout` seconds, the handler terminates.
- If more than `delayed_queue_size` rows are pending in a specific handler queue, the thread requesting `INSERT DELAYED` waits until there is room in the queue. This is done to ensure that `mysqld` does not use all memory for the delayed memory queue.
- The handler thread shows up in the MySQL process list with `delayed_insert` in the `Command` column. It is killed if you execute a `FLUSH TABLES` statement or kill it with `KILL thread_id`. However, before exiting, it first stores all queued rows into the table. During this time it does not accept any new `INSERT` statements from other threads. If you execute an `INSERT DELAYED` statement after this, a new handler thread is created.

Note that this means that `INSERT DELAYED` statements have higher priority than normal `INSERT` statements if there is an `INSERT DELAYED` handler running. Other update statements have to wait until the `INSERT DELAYED` queue is empty, someone terminates the handler thread (with `KILL thread_id`), or someone executes a `FLUSH TABLES`.

- The following status variables provide information about `INSERT DELAYED` statements:

Status Variable	Meaning
<code>Delayed_insert_threads</code>	Number of handler threads
<code>Delayed_writes</code>	Number of rows written with <code>INSERT DELAYED</code>
<code>Not_flushed_delayed_rows</code>	Number of rows waiting to be written

You can view these variables by issuing a `SHOW STATUS` statement or by executing a `mysqladmin extended-status` command.

12.2.4.3. `INSERT ... ON DUPLICATE KEY UPDATE` Syntax

If you specify `ON DUPLICATE KEY UPDATE`, and a row is inserted that would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row is performed. For example, if column `a` is declared as `UNIQUE` and contains the value `1`, the following two statements have identical effect:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE c=c+1;

UPDATE table SET c=c+1 WHERE a=1;
```

The rows-affected value is 1 if the row is inserted as a new record and 2 if an existing record is updated.

If column `b` is also unique, the `INSERT` is equivalent to this `UPDATE` statement instead:

```
UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

If `a=1 OR b=2` matches several rows, only *one* row is updated. In general, you should try to avoid using an `ON DUPLICATE KEY` clause on tables with multiple unique indexes.

You can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the `INSERT ... UPDATE` statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of `col_name` that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` function is meaningful only in `INSERT ... UPDATE` statements and returns `NULL` otherwise. Example:

```
INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

That statement is identical to the following two statements:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE c=3;
INSERT INTO table (a,b,c) VALUES (4,5,6)
ON DUPLICATE KEY UPDATE c=9;
```

If a table contains an `AUTO_INCREMENT` column and `INSERT ... UPDATE` inserts a row, the `LAST_INSERT_ID()` function returns the `AUTO_INCREMENT` value. If the statement updates a row instead, `LAST_INSERT_ID()` is not meaningful. However, you can work around this by using `LAST_INSERT_ID(expr)`. Suppose that `id` is the `AUTO_INCREMENT` column. To make `LAST_INSERT_ID()` meaningful for updates, insert rows as follows:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE id=LAST_INSERT_ID(id), c=3;
```

The `DELAYED` option is ignored when you use `ON DUPLICATE KEY UPDATE`.

12.2.5. LOAD DATA INFILE Syntax

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
[REPLACE | IGNORE]
INTO TABLE tbl_name
[CHARACTER SET charset_name]
[FIELDS
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
]
[LINES
  [STARTING BY 'string']
  [TERMINATED BY 'string']
]
[IGNORE number LINES]
[(col_name_or_user_var,...)]
[SET col_name = expr,...]
```

The `LOAD DATA INFILE` statement reads rows from a text file into a table at a very high speed. The filename must be given as a literal string.

`LOAD DATA INFILE` is the complement of `SELECT ... INTO OUTFILE`. (See [Section 12.2.7, “SELECT Syntax”](#).) To write data from a table to a file, use `SELECT ... INTO OUTFILE`. To read the file back into a table, use `LOAD DATA INFILE`. The syntax of the `FIELDS` and `LINES` clauses is the same for both statements. Both clauses are optional, but `FIELDS` must precede `LINES` if both are specified.

For more information about the efficiency of `INSERT` versus `LOAD DATA INFILE` and speeding up `LOAD DATA INFILE`, see [Section 6.2.17, “Speed of INSERT Statements”](#).

The character set indicated by the `character_set_database` system variable is used to interpret the information in the file. `SET NAMES` and the setting of `character_set_client` do not affect interpretation of input. Beginning with MySQL 5.1.17, if the contents of the input file use a character set that differs from the default, it is possible (and usually preferable) to use the `CHARACTER SET` clause to specify the character set of the file.

`LOAD DATA INFILE` interprets all fields in the file as having the same character set, regardless of the data types of the columns into which field values are loaded. For proper interpretation of file contents, you must ensure that it was written with the correct character set. For example, if you write a data file with `mysqldump -T` or by issuing a `SELECT ... INTO OUTFILE` statement in `mysql`, be sure to use a `--default-character-set` option with `mysqldump` or `mysql` so that output is written in the character set to be used when the file is loaded with `LOAD DATA INFILE`.

Note that it is currently not possible to load data files that use the `ucs2` character set.

As of MySQL 5.1.6, the `character_set_filesystem` system variable controls the interpretation of the filename.

You can also load data files by using the `mysqlimport` utility; it operates by sending a `LOAD DATA INFILE` statement to the server. The `--local` option causes `mysqlimport` to read data files from the client host. You can specify the `--compress` option to get better performance over slow networks if the client and server support the compressed protocol. See [Section 7.15, “mysqlimport — A Data Import Program”](#).

If you use `LOW_PRIORITY`, execution of the `LOAD DATA` statement is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (`MyISAM`, `MEMORY`, `MERGE`).

If you specify **CONCURRENT** with a **MyISAM** table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other threads can retrieve data from the table while **LOAD DATA** is executing. Using this option affects the performance of **LOAD DATA** a bit, even if no other thread is using the table at the same time.

The **LOCAL** keyword, if specified, is interpreted with respect to the client end of the connection:

- If **LOCAL** is specified, the file is read by the client program on the client host and sent to the server. The file can be given as a full pathname to specify its exact location. If given as a relative pathname, the name is interpreted relative to the directory in which the client program was started.
- If **LOCAL** is not specified, the file must be located on the server host and is read directly by the server. The server uses the following rules to locate the file:
 - If the filename is an absolute pathname, the server uses it as given.
 - If the filename is a relative pathname with one or more leading components, the server searches for the file relative to the server's data directory.
 - If a filename with no leading components is given, the server looks for the file in the database directory of the default database.

Note that, in the non-**LOCAL** case, these rules mean that a file named as `./myfile.txt` is read from the server's data directory, whereas the file named as `myfile.txt` is read from the database directory of the default database. For example, if **db1** is the default database, the following **LOAD DATA** statement reads the file `data.txt` from the database directory for **db1**, even though the statement explicitly loads the file into a table in the **db2** database:

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Windows pathnames are specified using forward slashes rather than backslashes. If you do use backslashes, you must double them.

For security reasons, when reading text files located on the server, the files must either reside in the database directory or be readable by all. Also, to use **LOAD DATA INFILE** on server files, you must have the **FILE** privilege. See [Section 5.7.3, “Privileges Provided by MySQL”](#).

Using **LOCAL** is a bit slower than letting the server access the files directly, because the contents of the file must be sent over the connection by the client to the server. On the other hand, you do not need the **FILE** privilege to load local files.

LOCAL works only if your server and your client both have been enabled to allow it. For example, if `mysqld` was started with `-local-infile=0`, **LOCAL** does not work. See [Section 5.6.4, “Security Issues with LOAD DATA LOCAL”](#).

On Unix, if you need **LOAD DATA** to read from a pipe, you can use the following technique (here we load the listing of the `/` directory into a table):

```
mkfifo /mysql/db/x/x
chmod 666 /mysql/db/x/x
find / -ls > /mysql/db/x/x &
mysql -e "LOAD DATA INFILE 'x' INTO TABLE x" x
```

Note that you must run the command that generates the data to be loaded and the `mysql` commands either on separate terminals, or run the data generation process in the background (as shown in the preceding example). If you do not do this, the pipe will block until data is read by the `mysql` process.

The **REPLACE** and **IGNORE** keywords control handling of input rows that duplicate existing rows on unique key values:

- If you specify **REPLACE**, input rows replace existing rows. In other words, rows that have the same value for a primary key or unique index as an existing row. See [Section 12.2.6, “REPLACE Syntax”](#).
- If you specify **IGNORE**, input rows that duplicate an existing row on a unique key value are skipped. If you do not specify either option, the behavior depends on whether the **LOCAL** keyword is specified. Without **LOCAL**, an error occurs when a duplicate key value is found, and the rest of the text file is ignored. With **LOCAL**, the default behavior is the same as if **IGNORE** is specified; this is because the server has no way to stop transmission of the file in the middle of the operation.

If you want to ignore foreign key constraints during the load operation, you can issue a **SET FOREIGN_KEY_CHECKS=0** statement

before executing `LOAD DATA`.

If you use `LOAD DATA INFILE` on an empty `MyISAM` table, all non-unique indexes are created in a separate batch (as for `REPAIR TABLE`). Normally, this makes `LOAD DATA INFILE` much faster when you have many indexes. In some extreme cases, you can create the indexes even faster by turning them off with `ALTER TABLE ... DISABLE KEYS` before loading the file into the table and using `ALTER TABLE ... ENABLE KEYS` to re-create the indexes after loading the file. See [Section 6.2.17, “Speed of INSERT Statements”](#).

For both the `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements, the syntax of the `FIELDS` and `LINES` clauses is the same. Both clauses are optional, but `FIELDS` must precede `LINES` if both are specified.

If you specify a `FIELDS` clause, each of its subclauses (`TERMINATED BY`, `[OPTIONALLY] ENCLOSED BY`, and `ESCAPED BY`) is also optional, except that you must specify at least one of them.

If you specify no `FIELDS` clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

If you specify no `LINES` clause, the defaults are the same as if you had written this:

```
LINES TERMINATED BY '\n' STARTING BY ''
```

In other words, the defaults cause `LOAD DATA INFILE` to act as follows when reading input:

- Look for line boundaries at newlines.
- Do not skip over any line prefix.
- Break lines into fields at tabs.
- Do not expect fields to be enclosed within any quoting characters.
- Interpret occurrences of tab, newline, or `\` preceded by `\` as literal characters that are part of field values.

Conversely, the defaults cause `SELECT ... INTO OUTFILE` to act as follows when writing output:

- Write tabs between fields.
- Do not enclose fields within any quoting characters.
- Use `\` to escape instances of tab, newline, or `\` that occur within field values.
- Write newlines at the ends of lines.

Backslash is the MySQL escape character within strings, so to write `FIELDS ESCAPED BY '\\'`, you must specify two backslashes for the value to be interpreted as a single backslash.

Note

If you have generated the text file on a Windows system, you might have to use `LINES TERMINATED BY '\r\n'` to read the file properly, because Windows programs typically use two characters as a line terminator. Some programs, such as `WordPad`, might use `\r` as a line terminator when writing files. To read such files, use `LINES TERMINATED BY '\r'`.

If all the lines you want to read in have a common prefix that you want to ignore, you can use `LINES STARTING BY 'prefix_string'` to skip over the prefix, *and anything before it*. If a line does not include the prefix, the entire line is skipped. Suppose that you issue the following statement:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
FIELDS TERMINATED BY ',' LINES STARTING BY 'xxx';
```

If the data file looks like this:


```
xxx"abc",1
something xxx"def",2
"ghi",3
```

The resulting rows will be ("abc", 1) and ("def", 2). The third row in the file is skipped because it does not contain the prefix.

The **IGNORE number LINES** option can be used to ignore lines at the start of the file. For example, you can use **IGNORE 1 LINES** to skip over an initial header line containing column names:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;
```

When you use **SELECT ... INTO OUTFILE** in tandem with **LOAD DATA INFILE** to write data from a database into a file and then read the file back into the database later, the field- and line-handling options for both statements must match. Otherwise, **LOAD DATA INFILE** will not interpret the contents of the file properly. Suppose that you use **SELECT ... INTO OUTFILE** to write a file with fields delimited by commas:

```
SELECT * INTO OUTFILE 'data.txt'
  FIELDS TERMINATED BY ','
FROM table2;
```

To read the comma-delimited file back in, the correct statement would be:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY ',';
```

If instead you tried to read in the file with the statement shown following, it wouldn't work because it instructs **LOAD DATA INFILE** to look for tabs between fields:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

LOAD DATA INFILE can be used to read files obtained from external sources. For example, many programs can export data in comma-separated values (CSV) format, such that lines have fields separated by commas and enclosed within double quotes. If lines in such a file are terminated by newlines, the statement shown here illustrates the field- and line-handling options you would use to load the file:

```
LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\n';
```

If the input values are not necessarily enclosed within quotes, use **OPTIONALLY** before the **ENCLOSED BY** keywords.

Any of the field- or line-handling options can specify an empty string (''). If not empty, the **FIELDS [OPTIONALLY] ENCLOSED BY** and **FIELDS ESCAPED BY** values must be a single character. The **FIELDS TERMINATED BY**, **LINES STARTING BY**, and **LINES TERMINATED BY** values can be more than one character. For example, to write lines that are terminated by carriage return/linefeed pairs, or to read a file containing such lines, specify a **LINES TERMINATED BY '\r\n'** clause.

To read a file containing jokes that are separated by lines consisting of **%%**, you can do this

```
CREATE TABLE jokes
(a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
 joke TEXT NOT NULL);
LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
  FIELDS TERMINATED BY ','
  LINES TERMINATED BY '\n%%\n' (joke);
```

FIELDS [OPTIONALLY] ENCLOSED BY controls quoting of fields. For output (**SELECT ... INTO OUTFILE**), if you omit the word **OPTIONALLY**, all fields are enclosed by the **ENCLOSED BY** character. An example of such output (using a comma as the field delimiter) is shown here:

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

If you specify **OPTIONALLY**, the **ENCLOSED BY** character is used only to enclose values from columns that have a string data type (such as **CHAR**, **BINARY**, **TEXT**, or **ENUM**):

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Note that occurrences of the **ENCLOSED BY** character within a field value are escaped by prefixing them with the **ESCAPED BY** character. Also note that if you specify an empty **ESCAPED BY** value, it is possible to inadvertently generate output that cannot be read properly by **LOAD DATA INFILE**. For example, the preceding output just shown would appear as follows if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

For input, the **ENCLOSED BY** character, if present, is stripped from the ends of field values. (This is true regardless of whether **OPTIONALLY** is specified; **OPTIONALLY** has no effect on input interpretation.) Occurrences of the **ENCLOSED BY** character preceded by the **ESCAPED BY** character are interpreted as part of the current field value.

If the field begins with the **ENCLOSED BY** character, instances of that character are recognized as terminating a field value only if followed by the field or line **TERMINATED BY** sequence. To avoid ambiguity, occurrences of the **ENCLOSED BY** character within a field value can be doubled and are interpreted as a single instance of the character. For example, if **ENCLOSED BY '''** is specified, quotes are handled as shown here:

```
"The ""BIG"" boss" -> The "BIG" boss
The "BIG" boss    -> The "BIG" boss
The ""BIG"" boss  -> The "BIG" boss
```

FIELDS ESCAPED BY controls how to write or read special characters. If the **FIELDS ESCAPED BY** character is not empty, it is used to prefix the following characters on output:

- The **FIELDS ESCAPED BY** character
- The **FIELDS [OPTIONALLY] ENCLOSED BY** character
- The first character of the **FIELDS TERMINATED BY** and **LINES TERMINATED BY** values
- ASCII 0 (what is actually written following the escape character is ASCII '0', not a zero-valued byte)

If the **FIELDS ESCAPED BY** character is empty, no characters are escaped and **NULL** is output as **NULL**, not **\N**. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

For input, if the **FIELDS ESCAPED BY** character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. Some two-character sequences that are exceptions, where the first character is the escape character. These sequences are shown in the following table (using '\' for the escape character). The rules for **NULL** handling are described later in this section.

\0	An ASCII 0 (NUL) character
\b	A backspace character
\n	A newline (linefeed) character
\r	A carriage return character
\t	A tab character.
\Z	ASCII 26 (Control-Z)
\N	NULL

For more information about ‘\’-escape syntax, see [Section 8.1, “Literal Values”](#).

In certain cases, field- and line-handling options interact:

- If **LINES TERMINATED BY** is an empty string and **FIELDS TERMINATED BY** is non-empty, lines are also terminated with **FIELDS TERMINATED BY**.
- If the **FIELDS TERMINATED BY** and **FIELDS ENCLOSED BY** values are both empty (' '), a fixed-row (non-delimited) format is used. With fixed-row format, no delimiters are used between fields (but you can still have a line terminator). Instead, column values are read and written using a field width wide enough to hold all values in the field. For **TINYINT**, **SMALLINT**, **MEDIUMINT**, **INT**, and **BIGINT**, the field widths are 4, 6, 8, 11, and 20, respectively, no matter what the declared display width is.

LINES TERMINATED BY is still used to separate lines. If a line does not contain all fields, the rest of the columns are set to their default values. If you do not have a line terminator, you should set this to ' '. In this case, the text file must contain all fields for each row.

Fixed-row format also affects handling of **NULL** values, as described later. Note that fixed-size format does not work if you are using a multi-byte character set.

Handling of **NULL** values varies according to the **FIELDS** and **LINES** options in use:

- For the default **FIELDS** and **LINES** values, **NULL** is written as a field value of **\N** for output, and a field value of **\N** is read as **NULL** for input (assuming that the **ESCAPED BY** character is ‘\’).
- If **FIELDS ENCLOSED BY** is not empty, a field containing the literal word **NULL** as its value is read as a **NULL** value. This differs from the word **NULL** enclosed within **FIELDS ENCLOSED BY** characters, which is read as the string 'NULL'.
- If **FIELDS ESCAPED BY** is empty, **NULL** is written as the word **NULL**.
- With fixed-row format (which is used when **FIELDS TERMINATED BY** and **FIELDS ENCLOSED BY** are both empty), **NULL** is written as an empty string. Note that this causes both **NULL** values and empty strings in the table to be indistinguishable when written to the file because both are written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

An attempt to load **NULL** into a **NOT NULL** column causes assignment of the implicit default value for the column's data type and a warning, or an error in strict SQL mode. Implicit default values are discussed in [Section 10.1.4, “Data Type Default Values”](#).

Some cases are not supported by **LOAD DATA INFILE**:

- Fixed-size rows (**FIELDS TERMINATED BY** and **FIELDS ENCLOSED BY** both empty) and **BLOB** or **TEXT** columns.
- If you specify one separator that is the same as or a prefix of another, **LOAD DATA INFILE** cannot interpret the input properly. For example, the following **FIELDS** clause would cause problems:

```
FIELDS TERMINATED BY ' ' ENCLOSED BY ' '
```

- If **FIELDS ESCAPED BY** is empty, a field value that contains an occurrence of **FIELDS ENCLOSED BY** or **LINES TERMINATED BY** followed by the **FIELDS TERMINATED BY** value causes **LOAD DATA INFILE** to stop reading a field or line too early. This happens because **LOAD DATA INFILE** cannot properly determine where the field or line value ends.

The following example loads all columns of the **persondata** table:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

By default, when no column list is provided at the end of the **LOAD DATA INFILE** statement, input lines are expected to contain a field for each table column. If you want to load only some of a table's columns, specify a column list:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata (col1,col2,...);
```

You must also specify a column list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, MySQL cannot tell how to match input fields with table columns.

The column list can contain either column names or user variables. With user variables, the **SET** clause enables you to perform transformations on their values before assigning the result to columns.

User variables in the **SET** clause can be used in several ways. The following example uses the first input column directly for the value of `t1.column1`, and assigns the second input column to a user variable that is subjected to a division operation before being used for the value of `t1.column2`:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @var1)
  SET column2 = @var1/100;
```

The **SET** clause can be used to supply values not derived from the input file. The following statement sets `column3` to the current date and time:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, column2)
  SET column3 = CURRENT_TIMESTAMP;
```

You can also discard an input value by assigning it to a user variable and not assigning the variable to a table column:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @dummy, column2, @dummy, column3);
```

Use of the column/variable list and **SET** clause is subject to the following restrictions:

- Assignments in the **SET** clause should have only column names on the left hand side of assignment operators.
- You can use subqueries in the right hand side of **SET** assignments. A subquery that returns a value to be assigned to a column may be a scalar subquery only. Also, you cannot use a subquery to select from the table that is being loaded.
- Lines ignored by an **IGNORE** clause are not processed for the column/variable list or **SET** clause.
- User variables cannot be used when loading data with fixed-row format because user variables do not have a display width.

When processing an input line, **LOAD DATA** splits it into fields and uses the values according to the column/variable list and the **SET** clause, if they are present. Then the resulting row is inserted into the table. If there are **BEFORE INSERT** or **AFTER INSERT** triggers for the table, they are activated before or after inserting the row, respectively.

If an input line has too many fields, the extra fields are ignored and the number of warnings is incremented.

If an input line has too few fields, the table columns for which input fields are missing are set to their default values. Default value assignment is described in [Section 10.1.4, “Data Type Default Values”](#).

An empty field value is interpreted differently than if the field value is missing:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to 0.
- For date and time types, the column is set to the appropriate “zero” value for the type. See [Section 10.3, “Date and Time Types”](#).

These are the same values that result if you assign an empty string explicitly to a string, numeric, or date or time type explicitly in an **INSERT** or **UPDATE** statement.

TIMESTAMP columns are set to the current date and time only if there is a **NULL** value for the column (that is, `\N`) and the column is not declared to allow **NULL** values, or if the **TIMESTAMP** column's default value is the current timestamp and it is omitted from the field list when a field list is specified.

LOAD DATA INFILE regards all input as strings, so you cannot use numeric values for **ENUM** or **SET** columns the way you can with **INSERT** statements. All **ENUM** and **SET** values must be specified as strings.

BIT values cannot be loaded using binary notation (for example, `b'011010'`). To work around this, specify the values as regular integers and use the **SET** clause to convert them so that MySQL performs a numeric type conversion and loads them into the **BIT** column properly:

```
shell> cat /tmp/bit_test.txt
2
127
shell> mysql test
mysql> LOAD DATA INFILE '/tmp/bit_test.txt'
-> INTO TABLE bit_test (@var1) SET b= CAST(@var1 AS UNSIGNED);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Deleted: 0 Skipped: 0 Warnings: 0

mysql> SELECT BIN(b+0) FROM bit_test;
+-----+
| bin(b+0) |
+-----+
| 10      |
| 1111111 |
+-----+
2 rows in set (0.00 sec)
```

When the **LOAD DATA INFILE** statement finishes, it returns an information string in the following format:

```
Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
```

If you are using the C API, you can get information about the statement by calling the `mysql_info()` function. See [Section 25.2.3.35, “mysql_info\(\)”](#).

Warnings occur under the same circumstances as when values are inserted via the **INSERT** statement (see [Section 12.2.4, “INSERT Syntax”](#)), except that **LOAD DATA INFILE** also generates warnings when there are too few or too many fields in the input row. The warnings are not stored anywhere; the number of warnings can be used only as an indication of whether everything went well.

You can use **SHOW WARNINGS** to get a list of the first `max_error_count` warnings as information about what went wrong. See [Section 12.5.4.32, “SHOW WARNINGS Syntax”](#).

12.2.6. REPLACE Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name,...)]
VALUES ({expr | DEFAULT},...),(...),...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name
SET col_name={expr | DEFAULT}, ...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name,...)]
SELECT ...
```

REPLACE works exactly like **INSERT**, except that if an old row in the table has the same value as a new row for a **PRIMARY KEY** or a **UNIQUE** index, the old row is deleted before the new row is inserted. See [Section 12.2.4, “INSERT Syntax”](#).

REPLACE is a MySQL extension to the SQL standard. It either inserts, or *deletes* and inserts. For another MySQL extension to standard SQL — that either inserts or *updates* — see [Section 12.2.4.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”](#).

Note that unless the table has a **PRIMARY KEY** or **UNIQUE** index, using a **REPLACE** statement makes no sense. It becomes equivalent to **INSERT**, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values specified in the **REPLACE** statement. Any missing columns are set to their default values, just as happens for **INSERT**. You cannot refer to values from the current row and use them in the new row. If you use an assignment such as `SET col_name = col_name + 1`, the reference to the column name on the right hand side is treated as

`col_name`), so the assignment is equivalent to `SET col_name = DEFAULT(col_name) + 1`.

To use `REPLACE`, you must have both the `INSERT` and `DELETE` privileges for the table.

The `REPLACE` statement returns a count to indicate the number of rows affected. This is the sum of the rows deleted and inserted. If the count is 1 for a single-row `REPLACE`, a row was inserted and no rows were deleted. If the count is greater than 1, one or more old rows were deleted before the new row was inserted. It is possible for a single row to replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

The affected-rows count makes it easy to determine whether `REPLACE` only added a row or whether it also replaced any rows: Check whether the count is 1 (added) or greater (replaced).

If you are using the C API, the affected-rows count can be obtained using the `mysql_affected_rows()` function.

Currently, you cannot replace into a table and select from the same table in a subquery.

MySQL uses the following algorithm for `REPLACE` (and `LOAD DATA ... REPLACE`):

1. Try to insert the new row into the table
2. While the insertion fails because a duplicate-key error occurs for a primary key or unique index:
 - a. Delete from the table the conflicting row that has the duplicate key value
 - b. Try again to insert the new row into the table

12.2.7. SELECT Syntax

```
SELECT
  [ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr, ...
  [FROM table_references
  [WHERE where_condition]
  [GROUP BY {col_name | expr | position}
  [ASC | DESC], ... [WITH ROLLUP]]
  [HAVING where_condition]
  [ORDER BY {col_name | expr | position}
  [ASC | DESC], ...]
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]
  [PROCEDURE procedure_name(argument_list)]
  [INTO OUTFILE 'file_name' export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name]]
  [FOR UPDATE | LOCK IN SHARE MODE]]
```

`SELECT` is used to retrieve rows selected from one or more tables, and can include `UNION` statements and subqueries. See [Section 12.2.7.3, “UNION Syntax”](#), and [Section 12.2.8, “Subquery Syntax”](#).

The most commonly used clauses of `SELECT` statements are these:

- Each `select_expr` indicates a column that you want to retrieve. There must be at least one `select_expr`.
- `table_references` indicates the table or tables from which to retrieve rows. Its syntax is described in [Section 12.2.7.1, “JOIN Syntax”](#).
- The `WHERE` clause, if given, indicates the condition or conditions that rows must satisfy to be selected. `where_condition` is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no `WHERE` clause.

In the `WHERE` clause, you can use any of the functions and operators that MySQL supports, except for aggregate (summary) functions. See [Chapter 11, Functions and Operators](#).

`SELECT` can also be used to retrieve rows computed without reference to any table.

For example:

```
mysql> SELECT 1 + 1;
-> 2
```

You are allowed to specify **DUAL** as a dummy table name in situations where no tables are referenced:

```
mysql> SELECT 1 + 1 FROM DUAL;
-> 2
```

DUAL is purely for the convenience of people who require that all **SELECT** statements should have **FROM** and possibly other clauses. MySQL may ignore the clauses. MySQL does not require **FROM DUAL** if no tables are referenced.

In general, clauses used must be given in exactly the order shown in the syntax description. For example, a **HAVING** clause must come after any **GROUP BY** clause and before any **ORDER BY** clause. The exception is that the **INTO** clause can appear either as shown in the syntax description or immediately preceding the **FROM** clause.

- A *select_expr* can be given an alias using **AS alias_name**. The alias is used as the expression's column name and can be used in **GROUP BY**, **ORDER BY**, or **HAVING** clauses. For example:

```
SELECT CONCAT(last_name, ', ', first_name) AS full_name
FROM mytable ORDER BY full_name;
```

The **AS** keyword is optional when aliasing a *select_expr*. The preceding example could have been written like this:

```
SELECT CONCAT(last_name, ', ', first_name) full_name
FROM mytable ORDER BY full_name;
```

However, because the **AS** is optional, a subtle problem can occur if you forget the comma between two *select_expr* expressions: MySQL interprets the second as an alias name. For example, in the following statement, **columnb** is treated as an alias name:

```
SELECT columna columnb FROM mytable;
```

For this reason, it is good practice to be in the habit of using **AS** explicitly when specifying column aliases.

- It is not allowable to use a column alias in a **WHERE** clause, because the column value might not yet be determined when the **WHERE** clause is executed. See [Section B.1.5.4, “Problems with Column Aliases”](#).
- The **FROM table_references** clause indicates the table or tables from which to retrieve rows. If you name more than one table, you are performing a join. For information on join syntax, see [Section 12.2.7.1, “JOIN Syntax”](#). For each table specified, you can optionally specify an alias.

```
tbl_name [[AS] alias] [index_hint]
```

The use of index hints provides the optimizer with information about how to choose indexes during query processing. For a description of the syntax for specifying these hints, see [Section 12.2.7.2, “Index Hint Syntax”](#).

You can use **SET max_seeks_for_key=value** as an alternative way to force MySQL to prefer key scans instead of table scans. See [Section 5.2.3, “System Variables”](#).

- You can refer to a table within the default database as *tbl_name*, or as *db_name.tbl_name* to specify a database explicitly. You can refer to a column as *col_name*, *tbl_name.col_name*, or *db_name.tbl_name.col_name*. You need not specify a *tbl_name* or *db_name.tbl_name* prefix for a column reference unless the reference would be ambiguous. See [Section 8.2.1, “Identifier Qualifiers”](#), for examples of ambiguity that require the more explicit column reference forms.
- A table reference can be aliased using *tbl_name AS alias_name* or *tbl_name alias_name*:

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
WHERE t1.name = t2.name;

SELECT t1.name, t2.salary FROM employee t1, info t2
WHERE t1.name = t2.name;
```

- Columns selected for output can be referred to in **ORDER BY** and **GROUP BY** clauses using column names, column aliases, or column positions. Column positions are integers and begin with 1:

```
SELECT college, region, seed FROM tournament
ORDER BY region, seed;

SELECT college, region AS r, seed AS s FROM tournament
ORDER BY r, s;

SELECT college, region, seed FROM tournament
ORDER BY 2, 3;
```

To sort in reverse order, add the **DESC** (descending) keyword to the name of the column in the **ORDER BY** clause that you are sorting by. The default is ascending order; this can be specified explicitly using the **ASC** keyword.

If **ORDER BY** occurs within a subquery and also is applied in the outer query, the outermost **ORDER BY** takes precedence. For example, results for the following statement are sorted in descending order, not ascending order:

```
(SELECT ... ORDER BY a) ORDER BY a DESC;
```

Use of column positions is deprecated because the syntax has been removed from the SQL standard.

- If you use **GROUP BY**, output rows are sorted according to the **GROUP BY** columns as if you had an **ORDER BY** for the same columns. To avoid the overhead of sorting that **GROUP BY** produces, add **ORDER BY NULL**:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a ORDER BY NULL;
```

- MySQL extends the **GROUP BY** clause so that you can also specify **ASC** and **DESC** after columns named in the clause:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a DESC;
```

- MySQL extends the use of **GROUP BY** to allow selecting fields that are not mentioned in the **GROUP BY** clause. If you are not getting the results that you expect from your query, please read the description of **GROUP BY** found in [Section 11.12, “Functions and Modifiers for Use with GROUP BY Clauses”](#).
- GROUP BY** allows a **WITH ROLLUP** modifier. See [Section 11.12.2, “GROUP BY Modifiers”](#).
- The **HAVING** clause is applied nearly last, just before items are sent to the client, with no optimization. (**LIMIT** is applied after **HAVING**.)

The SQL standard requires that **HAVING** must reference only columns in the **GROUP BY** clause or columns used in aggregate functions. However, MySQL supports an extension to this behavior, and allows **HAVING** to refer to columns in the **SELECT** list and columns in outer subqueries as well.

If the **HAVING** clause refers to a column that is ambiguous, a warning occurs. In the following statement, **col2** is ambiguous because it is used as both an alias and a column name:

```
SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

Preference is given to standard SQL behavior, so if a **HAVING** column name is used both in **GROUP BY** and as an aliased column in the output column list, preference is given to the column in the **GROUP BY** column.

- Do not use **HAVING** for items that should be in the **WHERE** clause. For example, do not write the following:

```
SELECT col_name FROM tbl_name HAVING col_name > 0;
```

Write this instead:

```
SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- The **HAVING** clause can refer to aggregate functions, which the **WHERE** clause cannot:

```
SELECT user, MAX(salary) FROM users
GROUP BY user HAVING MAX(salary) > 10;
```


(This did not work in some older versions of MySQL.)

- MySQL allows duplicate column names. That is, there can be more than one *select_expr* with the same name. This is an extension to standard SQL. Because MySQL also allows **GROUP BY** and **HAVING** to refer to *select_expr* values, this can result in an ambiguity:

```
SELECT 12 AS a, a FROM t GROUP BY a;
```

In that statement, both columns have the name *a*. To ensure that the correct column is used for grouping, use different names for each *select_expr*.

- MySQL resolves unqualified column or alias references in **ORDER BY** clauses by searching in the *select_expr* values, then in the columns of the tables in the **FROM** clause. For **GROUP BY** or **HAVING** clauses, it searches the **FROM** clause before searching in the *select_expr* values. (For **GROUP BY** and **HAVING**, this differs from the pre-MySQL 5.0 behavior that used the same rules as for **ORDER BY**.)
- The **LIMIT** clause can be used to constrain the number of rows returned by the **SELECT** statement. **LIMIT** takes one or two numeric arguments, which must both be non-negative integer constants (except when using prepared statements).

With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1):

```
SELECT * FROM tbl LIMIT 5,10; # Retrieve rows 6-15
```

To retrieve all rows from a certain offset up to the end of the result set, you can use some large number for the second parameter. This statement retrieves all rows from the 96th row to the last:

```
SELECT * FROM tbl LIMIT 95,18446744073709551615;
```

With one argument, the value specifies the number of rows to return from the beginning of the result set:

```
SELECT * FROM tbl LIMIT 5; # Retrieve first 5 rows
```

In other words, **LIMIT row_count** is equivalent to **LIMIT 0, row_count**.

For prepared statements, you can use placeholders. The following statements will return one row from the *tbl* table:

```
SET @a=1;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';
EXECUTE STMT USING @a;
```

The following statements will return the second to sixth row from the *tbl* table:

```
SET @skip=1; SET @numrows=5;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
EXECUTE STMT USING @skip, @numrows;
```

For compatibility with PostgreSQL, MySQL also supports the **LIMIT row_count OFFSET offset** syntax.

If **LIMIT** occurs within a subquery and also is applied in the outer query, the outermost **LIMIT** takes precedence. For example, the following statement produces two rows, not one:

```
(SELECT ... LIMIT 1) LIMIT 2;
```

- The **SELECT ... INTO OUTFILE 'file_name'** form of **SELECT** writes the selected rows to a file. The file is created on the server host, so you must have the **FILE** privilege to use this syntax. *file_name* cannot be an existing file, which among other things prevents files such as */etc/passwd* and database tables from being destroyed. As of MySQL 5.1.6, the **character_set_filesystem** system variable controls the interpretation of the filename.

The **SELECT ... INTO OUTFILE** statement is intended primarily to let you very quickly dump a table to a text file on the server machine. If you want to create the resulting file on some client host other than the server host, you cannot use **SELECT ... INTO OUTFILE**. In that case, you should instead use a command such as **mysql -e "SELECT ..." > file_name** to generate the file on the client host.

`SELECT ... INTO OUTFILE` is the complement of `LOAD DATA INFILE`; the syntax for the *export_options* part of the statement consists of the same `FIELDS` and `LINES` clauses that are used with the `LOAD DATA INFILE` statement. See [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).

`FIELDS ESCAPED BY` controls how to write special characters. If the `FIELDS ESCAPED BY` character is not empty, it is used as a prefix that precedes following characters on output:

- The `FIELDS ESCAPED BY` character
- The `FIELDS [OPTIONALLY] ENCLOSED BY` character
- The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values
- ASCII NUL (the zero-valued byte; what is actually written following the escape character is ASCII ‘0’, not a zero-valued byte)

The `FIELDS TERMINATED BY`, `ENCLOSED BY`, `ESCAPED BY`, or `LINES TERMINATED BY` characters *must* be escaped so that you can read the file back in reliably. ASCII NUL is escaped to make it easier to view with some paggers.

The resulting file does not have to conform to SQL syntax, so nothing else need be escaped.

If the `FIELDS ESCAPED BY` character is empty, no characters are escaped and `NULL` is output as `NULL`, not `\N`. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

Here is an example that produces a file in the comma-separated values (CSV) format used by many programs:

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM test_table;
```

- If you use `INTO DUMPFILE` instead of `INTO OUTFILE`, MySQL writes only one row into the file, without any column or line termination and without performing any escape processing. This is useful if you want to store a `BLOB` value in a file.
- The `INTO` clause can name a list of one or more variables, which can be user-defined variables, or parameters or local variables within a stored function or procedure body. The selected values are assigned to the variables. The number of variables must match the number of columns.

Within a stored routine, the variables can be routine parameters or local variables. See [Section 19.2.7.3, “SELECT ... INTO Statement”](#).

Note

Any file created by `INTO OUTFILE` or `INTO DUMPFILE` is writable by all users on the server host. The reason for this is that the MySQL server cannot create a file that is owned by anyone other than the user under whose account it is running. (You should *never* run `mysqld` as `root` for this and other reasons.) The file thus must be world-writable so that you can manipulate its contents.

- The `SELECT` syntax description at the beginning this section shows the `INTO` clause near the end of the statement. It is also possible to use `INTO OUTFILE` or `INTO DUMPFILE` immediately preceding the `FROM` clause.
- A `PROCEDURE` clause names a procedure that should process the data in the result set. For an example, see [Section 28.4.1, “Procedure Analyse”](#).
- If you use `FOR UPDATE` with a storage engine that uses page or row locks, rows examined by the query are write-locked until the end of the current transaction. Using `LOCK IN SHARE MODE` sets a shared lock that allows other transactions to read the examined rows but not to update or delete them. See [Section 13.5.10.5, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”](#).

Following the `SELECT` keyword, you can use a number of options that affect the operation of the statement.

The `ALL`, `DISTINCT`, and `DISTINCTROW` options specify whether duplicate rows should be returned. If none of these options are given, the default is `ALL` (all matching rows are returned). `DISTINCT` and `DISTINCTROW` are synonyms and specify removal of duplic-

ate rows from the result set.

HIGH_PRIORITY, **STRAIGHT_JOIN**, and options beginning with **SQL_** are MySQL extensions to standard SQL.

- **HIGH_PRIORITY** gives the **SELECT** higher priority than a statement that updates a table. You should use this only for queries that are very fast and must be done at once. A **SELECT HIGH_PRIORITY** query that is issued while the table is locked for reading runs even if there is an update statement waiting for the table to be free. This affects only storage engines that use only table-level locking (**MyISAM**, **MEMORY**, **MERGE**).

HIGH_PRIORITY cannot be used with **SELECT** statements that are part of a **UNION**.

- **STRAIGHT_JOIN** forces the optimizer to join the tables in the order in which they are listed in the **FROM** clause. You can use this to speed up a query if the optimizer joins the tables in non-optimal order. See [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#). **STRAIGHT_JOIN** also can be used in the *table_references* list. See [Section 12.2.7.1, “JOIN Syntax”](#).
- **SQL_BIG_RESULT** can be used with **GROUP BY** or **DISTINCT** to tell the optimizer that the result set has many rows. In this case, MySQL directly uses disk-based temporary tables if needed, and prefers sorting to using a temporary table with a key on the **GROUP BY** elements.
- **SQL_BUFFER_RESULT** forces the result to be put into a temporary table. This helps MySQL free the table locks early and helps in cases where it takes a long time to send the result set to the client.
- **SQL_SMALL_RESULT** can be used with **GROUP BY** or **DISTINCT** to tell the optimizer that the result set is small. In this case, MySQL uses fast temporary tables to store the resulting table instead of using sorting. This should not normally be needed.
- **SQL_CALC_FOUND_ROWS** tells MySQL to calculate how many rows there would be in the result set, disregarding any **LIMIT** clause. The number of rows can then be retrieved with **SELECT FOUND_ROWS()**. See [Section 11.11.3, “Information Functions”](#).
- **SQL_CACHE** tells MySQL to store the query result in the query cache if you are using a *query_cache_type* value of 2 or **DEMAND**. For a query that uses **UNION** or subqueries, this option effects any **SELECT** in the query. See [Section 6.5.4, “The MySQL Query Cache”](#).
- **SQL_NO_CACHE** tells MySQL not to store the query result in the query cache. See [Section 6.5.4, “The MySQL Query Cache”](#). For a query that uses **UNION** or subqueries, this option effects any **SELECT** in the query.

12.2.7.1. JOIN Syntax

MySQL supports the following **JOIN** syntaxes for the *table_references* part of **SELECT** statements and multiple-table **DELETE** and **UPDATE** statements:

```

table_references:
    table_reference [, table_reference] ...

table_reference:
    table_factor
  | join_table

table_factor:
    tbl_name [[AS] alias] [index_hint_list]
  | ( table_references )
  | { OJ table_reference LEFT OUTER JOIN table_reference
      ON conditional_expr }

join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference STRAIGHT_JOIN table_factor
  | table_reference STRAIGHT_JOIN table_factor ON condition
  | table_reference LEFT [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [LEFT [OUTER]] JOIN table_factor
  | table_reference RIGHT [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [RIGHT [OUTER]] JOIN table_factor

join_condition:
    ON conditional_expr
  | USING (column_list)

index_hint_list:
    index_hint [, index_hint] ...

index_hint:
    USE {INDEX|KEY}
      [{FOR {JOIN|ORDER BY|GROUP BY}}] ([index_list])
  | IGNORE {INDEX|KEY}
      [{FOR {JOIN|ORDER BY|GROUP BY}}] (index_list)

```

```
| FORCE {INDEX|KEY}
  [{FOR {JOIN|ORDER BY|GROUP BY}}] (index_list)
index_list:
  index_name [, index_name] ...
```

A table reference is also known as a join expression.

The syntax of *table_factor* is extended in comparison with the SQL Standard. The latter accepts only *table_reference*, not a list of them inside a pair of parentheses.

This is a conservative extension if we consider each comma in a list of *table_reference* items as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                      ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                      ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

In MySQL, **CROSS JOIN** is a syntactic equivalent to **INNER JOIN** (they can replace each other). In standard SQL, they are not equivalent. **INNER JOIN** is used with an **ON** clause, **CROSS JOIN** is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. MySQL also supports nested joins (see [Section 6.2.9, “Nested Join Optimization”](#)).

Index hints can be specified to affect how the MySQL optimizer makes use of indexes. For more information, see [Section 12.2.7.2, “Index Hint Syntax”](#).

The following list describes general factors to take into account when writing joins.

- A table reference can be aliased using *tbl_name AS alias_name* or *tbl_name alias_name*:

```
SELECT t1.name, t2.salary
  FROM employee AS t1 INNER JOIN info AS t2 ON t1.name = t2.name;

SELECT t1.name, t2.salary
  FROM employee t1 INNER JOIN info t2 ON t1.name = t2.name;
```

- INNER JOIN** and **,** (comma) are semantically equivalent in the absence of a join condition: both produce a Cartesian product between the specified tables (that is, each and every row in the first table is joined to each and every row in the second table).

However, the precedence of the comma operator is less than of **INNER JOIN**, **CROSS JOIN**, **LEFT JOIN**, and so on. If you mix comma joins with the other join types when there is a join condition, an error of the form **Unknown column 'col_name' in 'on clause'** may occur. Information about dealing with this problem is given later in this section.

- The **ON** conditional is any conditional expression of the form that can be used in a **WHERE** clause. Generally, you should use the **ON** clause for conditions that specify how to join tables, and the **WHERE** clause to restrict which rows you want in the result set.
- If there is no matching row for the right table in the **ON** or **USING** part in a **LEFT JOIN**, a row with all columns set to **NULL** is used for the right table. You can use this fact to find rows in a table that have no counterpart in another table:

```
SELECT left_tbl.*
  FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id
 WHERE right_tbl.id IS NULL;
```

This example finds all rows in *left_tbl* with an *id* value that is not present in *right_tbl* (that is, all rows in *left_tbl* with no corresponding row in *right_tbl*). This assumes that *right_tbl.id* is declared **NOT NULL**. See [Section 6.2.8, “LEFT JOIN and RIGHT JOIN Optimization”](#).

- The **USING(column_list)** clause names a list of columns that must exist in both tables. If tables *a* and *b* both contain columns *c1*, *c2*, and *c3*, the following join compares corresponding columns from the two tables:

```
a LEFT JOIN b USING (c1,c2,c3)
```

- The **NATURAL [LEFT] JOIN** of two tables is defined to be semantically equivalent to an **INNER JOIN** or a **LEFT JOIN** with a **USING** clause that names all columns that exist in both tables.
- **RIGHT JOIN** works analogously to **LEFT JOIN**. To keep code portable across databases, it is recommended that you use **LEFT JOIN** instead of **RIGHT JOIN**.
- The **{ OJ ... LEFT OUTER JOIN ... }** syntax shown in the join syntax description exists only for compatibility with ODBC. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.
- **STRAIGHT_JOIN** is similar to **JOIN**, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer puts the tables in the wrong order.

Some join examples:

```
SELECT * FROM table1, table2;
SELECT * FROM table1 INNER JOIN table2 ON table1.id=table2.id;
SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;
SELECT * FROM table1 LEFT JOIN table2 USING (id);
SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id
LEFT JOIN table3 ON table2.id=table3.id;
```

Join Processing Changes in MySQL 5.0.12

Note

Natural joins and joins with **USING**, including outer join variants, are processed according to the SQL:2003 standard. The goal was to align the syntax and semantics of MySQL with respect to **NATURAL JOIN** and **JOIN ... USING** according to SQL:2003. However, these changes in join processing can result in different output columns for some joins. Also, some queries that appeared to work correctly in older versions (prior to 5.0.12) must be rewritten to comply with the standard.

These changes have five main aspects:

- The way that MySQL determines the result columns of **NATURAL** or **USING** join operations (and thus the result of the entire **FROM** clause).
- Expansion of **SELECT *** and **SELECT tbl_name.*** into a list of selected columns.
- Resolution of column names in **NATURAL** or **USING** joins.
- Transformation of **NATURAL** or **USING** joins into **JOIN ... ON**.
- Resolution of column names in the **ON** condition of a **JOIN ... ON**.

The following list provides more detail about several effects of current join processing versus join processing in older versions. The term “previously” means “prior to MySQL 5.0.12.”

- The columns of a **NATURAL** join or a **USING** join may be different from previously. Specifically, redundant output columns no longer appear, and the order of columns for **SELECT *** expansion may be different from before.

Consider this set of statements:

```
CREATE TABLE t1 (i INT, j INT);
CREATE TABLE t2 (k INT, j INT);
INSERT INTO t1 VALUES(1,1);
INSERT INTO t2 VALUES(1,1);
SELECT * FROM t1 NATURAL JOIN t2;
SELECT * FROM t1 JOIN t2 USING (j);
```

Previously, the statements produced this output:

i	j	k	j
1	1	1	1
i	j	k	j
1	1	1	1

In the first **SELECT** statement, column **j** appears in both tables and thus becomes a join column, so, according to standard SQL, it should appear only once in the output, not twice. Similarly, in the second **SELECT** statement, column **j** is named in the **USING** clause and should appear only once in the output, not twice. But in both cases, the redundant column is not eliminated. Also, the order of the columns is not correct according to standard SQL.

Now the statements produce this output:

j	i	k
1	1	1
j	i	k
1	1	1

The redundant column is eliminated and the column order is correct according to standard SQL:

- First, coalesced common columns of the two joined tables, in the order in which they occur in the first table
- Second, columns unique to the first table, in order in which they occur in that table
- Third, columns unique to the second table, in order in which they occur in that table

The single result column that replaces two common columns is defined via the coalesce operation. That is, for two **t1.a** and **t2.a** the resulting single join column **a** is defined as **a = COALESCE(t1.a, t2.a)**, where:

```
COALESCE(x, y) = (CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END)
```

If the join operation is any other join, the result columns of the join consists of the concatenation of all columns of the joined tables. This is the same as previously.

A consequence of the definition of coalesced columns is that, for outer joins, the coalesced column contains the value of the non-**NULL** column if one of the two columns is always **NULL**. If neither or both columns are **NULL**, both common columns have the same value, so it doesn't matter which one is chosen as the value of the coalesced column. A simple way to interpret this is to consider that a coalesced column of an outer join is represented by the common column of the inner table of a **JOIN**. Suppose that the tables **t1(a,b)** and **t2(a,c)** have the following contents:

t1	t2
1 x	2 z
2 y	3 w

Then:

```
mysql> SELECT * FROM t1 NATURAL LEFT JOIN t2;
```

a	b	c
1	x	NULL
2	y	z

Here column **a** contains the values of **t1.a**.

```
mysql> SELECT * FROM t1 NATURAL RIGHT JOIN t2;
```

a	c	b

2	z	y
3	w	NULL

Here column `a` contains the values of `t2.a`.

Compare these results to the otherwise equivalent queries with `JOIN ... ON`:

```
mysql> SELECT * FROM t1 LEFT JOIN t2 ON (t1.a = t2.a);
```

a	b	a	c
1	x	NULL	NULL
2	y	2	z

```
mysql> SELECT * FROM t1 RIGHT JOIN t2 ON (t1.a = t2.a);
```

a	b	a	c
2	y	2	z
NULL	NULL	3	w

- Previously, a `USING` clause could be rewritten as an `ON` clause that compares corresponding columns. For example, the following two clauses were semantically identical:

```
a LEFT JOIN b USING (c1,c2,c3)
a LEFT JOIN b ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
```

Now the two clauses no longer are quite the same:

- With respect to determining which rows satisfy the join condition, both joins remain semantically identical.
- With respect to determining which columns to display for `SELECT *` expansion, the two joins are not semantically identical. The `USING` join selects the coalesced value of corresponding columns, whereas the `ON` join selects all columns from all tables. For the preceding `USING` join, `SELECT *` selects these values:

```
COALESCE(a.c1,b.c1), COALESCE(a.c2,b.c2), COALESCE(a.c3,b.c3)
```

For the `ON` join, `SELECT *` selects these values:

```
a.c1, a.c2, a.c3, b.c1, b.c2, b.c3
```

With an inner join, `COALESCE(a.c1,b.c1)` is the same as either `a.c1` or `b.c1` because both columns will have the same value. With an outer join (such as `LEFT JOIN`), one of the two columns can be `NULL`. That column will be omitted from the result.

- The evaluation of multi-way natural joins differs in a very important way that affects the result of `NATURAL` or `USING` joins and that can require query rewriting. Suppose that you have three tables `t1(a,b)`, `t2(c,b)`, and `t3(a,c)` that each have one row: `t1(1,2)`, `t2(10,2)`, and `t3(7,10)`. Suppose also that you have this `NATURAL JOIN` on the three tables:

```
SELECT ... FROM t1 NATURAL JOIN t2 NATURAL JOIN t3;
```

Previously, the left operand of the second join was considered to be `t2`, whereas it should be the nested join (`t1 NATURAL JOIN t2`). As a result, the columns of `t3` are checked for common columns only in `t2`, and, if `t3` has common columns with `t1`, these columns are not used as equi-join columns. Thus, previously, the preceding query was transformed to the following equi-join:

```
SELECT ... FROM t1, t2, t3
WHERE t1.b = t2.b AND t2.c = t3.c;
```

That join is missing one more equi-join predicate (`t1.a = t3.a`). As a result, it produces one row, not the empty result that it should. The correct equivalent query is this:

```
SELECT ... FROM t1, t2, t3
WHERE t1.b = t2.b AND t2.c = t3.c AND t1.a = t3.a;
```

If you require the same query result in current versions of MySQL as in older versions, rewrite the natural join as the first equi-join.

- Previously, the comma operator (,) and **JOIN** both had the same precedence, so the join expression `t1, t2 JOIN t3` was interpreted as `((t1, t2) JOIN t3)`. Now **JOIN** has higher precedence, so the expression is interpreted as `(t1, (t2 JOIN t3))`. This change affects statements that use an **ON** clause, because that clause can refer only to columns in the operands of the join, and the change in precedence changes interpretation of what those operands are.

Example:

```
CREATE TABLE t1 (i1 INT, j1 INT);
CREATE TABLE t2 (i2 INT, j2 INT);
CREATE TABLE t3 (i3 INT, j3 INT);
INSERT INTO t1 VALUES(1,1);
INSERT INTO t2 VALUES(1,1);
INSERT INTO t3 VALUES(1,1);
SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);
```

Previously, the **SELECT** was legal due to the implicit grouping of `t1, t2` as `(t1, t2)`. Now the **JOIN** takes precedence, so the operands for the **ON** clause are `t2` and `t3`. Because `t1.i1` is not a column in either of the operands, the result is an **Unknown column 't1.i1' in 'on clause'** error. To allow the join to be processed, group the first two tables explicitly with parentheses so that the operands for the **ON** clause are `(t1, t2)` and `t3`:

```
SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);
```

Alternatively, avoid the use of the comma operator and use **JOIN** instead:

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (t1.i1 = t3.i3);
```

This change also applies to statements that mix the comma operator with **INNER JOIN**, **CROSS JOIN**, **LEFT JOIN**, and **RIGHT JOIN**, all of which now have higher precedence than the comma operator.

- Previously, the **ON** clause could refer to columns in tables named to its right. Now an **ON** clause can refer only to its operands.

Example:

```
CREATE TABLE t1 (i1 INT);
CREATE TABLE t2 (i2 INT);
CREATE TABLE t3 (i3 INT);
SELECT * FROM t1 JOIN t2 ON (i1 = i3) JOIN t3;
```

Previously, the **SELECT** statement was legal. Now the statement fails with an **Unknown column 'i3' in 'on clause'** error because `i3` is a column in `t3`, which is not an operand of the **ON** clause. The statement should be rewritten as follows:

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (i1 = i3);
```

- Resolution of column names in **NATURAL** or **USING** joins is different than previously. For column names that are outside the **FROM** clause, MySQL now handles a superset of the queries compared to previously. That is, in cases when MySQL formerly issued an error that some column is ambiguous, the query now is handled correctly. This is due to the fact that MySQL now treats the common columns of **NATURAL** or **USING** joins as a single column, so when a query refers to such columns, the query compiler does not consider them as ambiguous.

Example:

```
SELECT * FROM t1 NATURAL JOIN t2 WHERE b > 1;
```

Previously, this query would produce an error **ERROR 1052 (23000): Column 'b' in where clause is ambiguous**. Now the query produces the correct result:

b	c	y
4	2	3

One extension of MySQL compared to the SQL:2003 standard is that MySQL allows you to qualify the common (coalesced)

columns of **NATURAL** or **USING** joins (just as previously), while the standard disallows that.

12.2.7.2. Index Hint Syntax

You can provide hints to give the optimizer information about how to choose indexes during query processing. [Section 12.2.7.1, “JOIN Syntax”](#), describes the general syntax for specifying tables in a **SELECT** statement. The syntax for an individual table, including that for index hints, looks like this:

```
tbl_name [[AS] alias] [index_hint_list]
index_hint_list:
    index_hint [, index_hint] ...
index_hint:
    USE {INDEX|KEY}
        [{FOR {JOIN|ORDER BY|GROUP BY}}] ([index_list])
    | IGNORE {INDEX|KEY}
        [{FOR {JOIN|ORDER BY|GROUP BY}}] (index_list)
    | FORCE {INDEX|KEY}
        [{FOR {JOIN|ORDER BY|GROUP BY}}] (index_list)
index_list:
    index_name [, index_name] ...
```

By specifying **USE INDEX (index_list)**, you can tell MySQL to use only one of the named indexes to find rows in the table. The alternative syntax **IGNORE INDEX (index_list)** can be used to tell MySQL to not use some particular index or indexes. These hints are useful if **EXPLAIN** shows that MySQL is using the wrong index from the list of possible indexes.

You can also use **FORCE INDEX**, which acts like **USE INDEX (index_list)** but with the addition that a table scan is assumed to be very expensive. In other words, a table scan is used only if there is no way to use one of the given indexes to find rows in the table.

USE KEY, **IGNORE KEY**, and **FORCE KEY** are synonyms for **USE INDEX**, **IGNORE INDEX**, and **FORCE INDEX**.

Each hint requires the names of *indexes*, not the names of columns. The name of a **PRIMARY KEY** is **PRIMARY**. To see the index names for a table, use **SHOW INDEX**.

Index hints do not work for **FULLTEXT** indexes.

Prior to MySQL 5.1.17, **USE INDEX**, **IGNORE INDEX**, and **FORCE INDEX** affect only which indexes are used when MySQL decides how to find rows in the table and how to process joins. They do not affect whether an index is used when resolving an **ORDER BY** or **GROUP BY** clause.

Examples:

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
WHERE col1=1 AND col2=2 AND col3=3;

SELECT * FROM table1 IGNORE INDEX (col3_index)
WHERE col1=1 AND col2=2 AND col3=3;
```

As of MySQL 5.1.17, the syntax for index hints is extended in the following ways:

- It is syntactically valid to specify an empty *index_list* for **USE INDEX**, which means “use no indexes.” Specifying an empty *index_list* for **FORCE INDEX** or **IGNORE INDEX** is a syntax error.
- You can specify the scope of a index hint by adding a **FOR** clause to the hint. This provides more fine-grained control over the optimizer's selection of an execution plan for various phases of query processing. To affect only the indexes used when MySQL decides how to find rows in the table and how to process joins, use **FOR JOIN**. To influence index usage for sorting or grouping rows, use **FOR ORDER BY** or **FOR GROUP BY**. (However, if there is a covering index for the table and it is used to access the table, the optimizer will ignore **IGNORE INDEX FOR {ORDER BY|GROUP BY}** hints that disable that index.)
- You can specify multiple index hints:

```
SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX FOR ORDER BY (i2) ORDER BY a;
```

It is not a error to name the same index in several hints (even within the same hint):

```
SELECT * FROM t1 USE INDEX (i1) USE INDEX (i1,i1);
```

However, it is an error to mix **USE INDEX** and **FORCE INDEX** for the same table:

```
SELECT * FROM t1 USE INDEX FOR JOIN (i1) FORCE INDEX FOR JOIN (i2);
```

The default scope of index hints also is changed as of MySQL 5.1.17. Formerly, index hints applied only to how indexes are used for retrieval of records and not during resolution of **ORDER BY** or **GROUP BY** clauses. As of 5.1.17, if you specify no **FOR** clause for an index hint, the hint by default applies to all parts of the statement. For example, this hint:

```
IGNORE INDEX (i1)
```

is equivalent to this combination of hints:

```
IGNORE INDEX FOR JOIN (i1)
IGNORE INDEX FOR ORDER BY (i1)
IGNORE INDEX FOR GROUP BY (i1)
```

To cause the server to use the older behavior for hint scope when no **FOR** clause is present (so that hints apply only to row retrieval), enable the **old** system variable at server startup. Take care about enabling this variable in a replication setup. With statement-based binary logging, having different modes for the master and slaves might lead to replication errors.

When index hints are processed, they are collected in a single list by type (**USE**, **FORCE**, **IGNORE**) and by scope (**FOR JOIN**, **FOR ORDER BY**, **FOR GROUP BY**). For example:

```
SELECT * FROM t1
  USE INDEX () IGNORE INDEX (i2) USE INDEX (i1) USE INDEX (i2);
```

is equivalent to:

```
SELECT * FROM t1
  USE INDEX (i1,i2) IGNORE INDEX (i2);
```

The index hints then are applied for each scope in the following order:

1. **{USE|FORCE} INDEX** is applied if present. (If not, the optimizer-determined set of indexes is used.)
2. **IGNORE INDEX** is applied over the result of the previous step. For example:

```
SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX (i2) USE INDEX (i2)
```

is equivalent to:

```
SELECT * FROM t1 USE INDEX (i1).
```

12.2.7.3. **UNION** Syntax

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

UNION is used to combine the result from multiple **SELECT** statements into a single result set.

The column names from the first **SELECT** statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each **SELECT** statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

If the data types of corresponding **SELECT** columns do not match, the types and lengths of the columns in the **UNION** result take into account the values retrieved by all of the **SELECT** statements. For example, consider the following:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+-----+
| REPEAT('a',1) |
+-----+
| a              |
| bbbbbbbbbbb   |
+-----+
```

(In some earlier versions of MySQL, only the type and length from the first **SELECT** would have been used and the second row would have been truncated to a length of 1.)

The **SELECT** statements are normal select statements, but with the following restrictions:

- Only the last **SELECT** statement can use **INTO OUTFILE**.
- **HIGH_PRIORITY** cannot be used with **SELECT** statements that are part of a **UNION**. If you specify it for the first **SELECT**, it has no effect. If you specify it for any subsequent **SELECT** statements, a syntax error results.

The default behavior for **UNION** is that duplicate rows are removed from the result. The optional **DISTINCT** keyword has no effect other than the default because it also specifies duplicate-row removal. With the optional **ALL** keyword, duplicate-row removal does not occur and the result includes all matching rows from all the **SELECT** statements.

You can mix **UNION ALL** and **UNION DISTINCT** in the same query. Mixed **UNION** types are treated such that a **DISTINCT** union overrides any **ALL** union to its left. A **DISTINCT** union can be produced explicitly by using **UNION DISTINCT** or implicitly by using **UNION** with no following **DISTINCT** or **ALL** keyword.

To use an **ORDER BY** or **LIMIT** clause to sort or limit the entire **UNION** result, parenthesize the individual **SELECT** statements and place the **ORDER BY** or **LIMIT** after the last one. The following example uses both clauses:

```
(SELECT a FROM t1 WHERE a=10 AND B=1)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

This kind of **ORDER BY** cannot use column references that include a table name (that is, names in *tbl_name.col_name* format). Instead, provide a column alias in the first **SELECT** statement and refer to the alias in the **ORDER BY**. (Alternatively, refer to the column in the **ORDER BY** using its column position. However, use of column positions is deprecated.)

Also, if a column to be sorted is aliased, the **ORDER BY** clause *must* refer to the alias, not the column name. The first of the following statements will work, but the second will fail with an **Unknown column 'a' in 'order clause'** error:

```
(SELECT a AS b FROM t) UNION (SELECT ... ) ORDER BY b;
(SELECT a AS b FROM t) UNION (SELECT ... ) ORDER BY a;
```

To apply **ORDER BY** or **LIMIT** to an individual **SELECT**, place the clause inside the parentheses that enclose the **SELECT**:

```
(SELECT a FROM t1 WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

Use of **ORDER BY** for individual **SELECT** statements implies nothing about the order in which the rows appear in the final result because **UNION** by default produces an unordered set of rows. If **ORDER BY** appears with **LIMIT**, it is used to determine the subset of the selected rows to retrieve for the **SELECT**, but does not necessarily affect the order of those rows in the final **UNION** result. If **ORDER BY** appears without **LIMIT** in a **SELECT**, it is optimized away because it will have no effect anyway.

To cause rows in a **UNION** result to consist of the sets of rows retrieved by each **SELECT** one after the other, select an additional column in each **SELECT** to use as a sort column and add an **ORDER BY** following the last **SELECT**:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col;
```

To additionally maintain sort order within individual **SELECT** results, add a secondary column to the **ORDER BY** clause:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
```

```
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col, col1a;
```

12.2.8. Subquery Syntax

A subquery is a [SELECT](#) statement within another statement.

Starting with MySQL 4.1, all subquery forms and operations that the SQL standard requires are supported, as well as a few features that are MySQL-specific.

Here is an example of a subquery:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

In this example, [SELECT * FROM t1 ...](#) is the *outer query* (or *outer statement*), and [\(SELECT column1 FROM t2\)](#) is the *subquery*. We say that the subquery is *nested* within the outer query, and in fact it is possible to nest subqueries within other subqueries, to a considerable depth. A subquery must always appear within parentheses.

The main advantages of subqueries are:

- They allow queries that are *structured* so that it is possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- They are, in many people's opinion, more readable than complex joins or unions. Indeed, it was the innovation of subqueries that gave people the original idea of calling the early SQL “Structured Query Language.”

Here is an example statement that shows the major points about subquery syntax as specified by the SQL standard and supported in MySQL:

```
DELETE FROM t1
WHERE s11 > ANY
  (SELECT COUNT(*) /* no hint */ FROM t2
  WHERE NOT EXISTS
    (SELECT * FROM t3
     WHERE ROW(5*t2.s1,77)=
      (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
       (SELECT * FROM t5) AS t5)));
```

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries. Subqueries that return a particular kind of result often can be used only in certain contexts, as described in the following sections.

There are few restrictions on the type of statements in which subqueries can be used. A subquery can contain any of the keywords or clauses that an ordinary [SELECT](#) can contain: [DISTINCT](#), [GROUP BY](#), [ORDER BY](#), [LIMIT](#), joins, index hints, [UNION](#) constructs, comments, functions, and so on.

One restriction is that a subquery's outer statement must be one of: [SELECT](#), [INSERT](#), [UPDATE](#), [DELETE](#), [SET](#), or [DO](#). Another restriction is that currently you cannot modify a table and select from the same table in a subquery. This applies to statements such as [DELETE](#), [INSERT](#), [REPLACE](#), [UPDATE](#), and (because subqueries can be used in the [SET](#) clause) [LOAD DATA INFILE](#).

A more comprehensive discussion of restrictions on subquery use, including performance issues for certain forms of subquery syntax, is given in [Section D.3, “Restrictions on Subqueries”](#).

12.2.8.1. The Subquery as Scalar Operand

In its simplest form, a subquery is a scalar subquery that returns a single value. A scalar subquery is a simple operand, and you can use it almost anywhere a single column value or literal is legal, and you can expect it to have those characteristics that all operands have: a data type, a length, an indication whether it can be [NULL](#), and so on. For example:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

The subquery in this [SELECT](#) returns a single value ('abcde') that has a data type of [CHAR](#), a length of 5, a character set and colla-

tion equal to the defaults in effect at **CREATE TABLE** time, and an indication that the value in the column can be **NULL**. In fact, almost all subqueries can be **NULL**. If the table used in the example were empty, the value of the subquery would be **NULL**.

There are a few contexts in which a scalar subquery cannot be used. If a statement allows only a literal value, you cannot use a subquery. For example, **LIMIT** requires literal integer arguments, and **LOAD DATA INFILE** requires a literal string filename. You cannot use subqueries to supply these values.

When you see examples in the following sections that contain the rather spartan construct **(SELECT column1 FROM t1)**, imagine that your own code contains much more diverse and complex constructions.

Suppose that we make two tables:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

Then perform a **SELECT**:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

The result is **2** because there is a row in **t2** containing a column **s1** that has a value of **2**.

A scalar subquery can be part of an expression, but remember the parentheses, even if the subquery is an operand that provides an argument for a function. For example:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

12.2.8.2. Comparisons Using Subqueries

The most common use of a subquery is in the form:

```
non_subquery_operand comparison_operator (subquery)
```

Where *comparison_operator* is one of these operators:

```
= > < >= <= <>
```

For example:

```
... 'a' = (SELECT column1 FROM t1)
```

At one time the only legal place for a subquery was on the right side of a comparison, and you might still find some old DBMSs that insist on this.

Here is an example of a common-form subquery comparison that you cannot do with a join. It finds all the values in table **t1** that are equal to a maximum value in table **t2**:

```
SELECT column1 FROM t1
WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Here is another example, which again is impossible with a join because it involves aggregating for one of the tables. It finds all rows in table **t1** containing a value that occurs twice in a given column:

```
SELECT * FROM t1 AS t
WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

For a comparison performed with one of these operators, the subquery must return a scalar, with the exception that **=** can be used with row subqueries. See [Section 12.2.8.5, “Row Subqueries”](#).

12.2.8.3. Subqueries with **ANY**, **IN**, and **SOME**

Syntax:

```
operand comparison_operator ANY (subquery)
```

```
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

The **ANY** keyword, which must follow a comparison operator, means “return **TRUE** if the comparison is **TRUE** for **ANY** of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suppose that there is a row in table **t1** containing (**10**). The expression is **TRUE** if table **t2** contains (**21, 14, 7**) because there is a value **7** in **t2** that is less than **10**. The expression is **FALSE** if table **t2** contains (**20, 10**), or if table **t2** is empty. The expression is *unknown* if table **t2** contains (**NULL, NULL, NULL**).

When used with a subquery, the word **IN** is an alias for **= ANY**. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN (SELECT s1 FROM t2);
```

IN and **= ANY** are not synonyms when used with an expression list. **IN** can take an expression list, but **= ANY** cannot. See [Section 11.2.3, “Comparison Functions and Operators”](#).

NOT IN is not an alias for **<> ANY**, but for **<> ALL**. See [Section 12.2.8.4, “Subqueries with ALL”](#).

The word **SOME** is an alias for **ANY**. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Use of the word **SOME** is rare, but this example shows why it might be useful. To most people's ears, the English phrase “a is not equal to any b” means “there is no b which is equal to a,” but that is not what is meant by the SQL syntax. The syntax means “there is some b to which a is not equal.” Using **<> SOME** instead helps ensure that everyone understands the true meaning of the query.

12.2.8.4. Subqueries with ALL

Syntax:

```
operand comparison_operator ALL (subquery)
```

The word **ALL**, which must follow a comparison operator, means “return **TRUE** if the comparison is **TRUE** for **ALL** of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suppose that there is a row in table **t1** containing (**10**). The expression is **TRUE** if table **t2** contains (**-5, 0, +5**) because **10** is greater than all three values in **t2**. The expression is **FALSE** if table **t2** contains (**12, 6, NULL, -100**) because there is a single value **12** in table **t2** that is greater than **10**. The expression is *unknown* (that is, **NULL**) if table **t2** contains (**0, NULL, 1**).

Finally, if table **t2** is empty, the result is **TRUE**. So, the following statement is **TRUE** when table **t2** is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

But this statement is **NULL** when table **t2** is empty:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

In addition, the following statement is **NULL** when table **t2** is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

In general, *tables containing NULL values* and *empty tables* are “edge cases.” When writing subquery code, always consider whether you have taken those two possibilities into account.

NOT IN is an alias for **<> ALL**. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
```

```
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

12.2.8.5. Row Subqueries

The discussion to this point has been of scalar or column subqueries; that is, subqueries that return a single value or a column of values. A *row subquery* is a subquery variant that returns a single row and can thus return more than one column value. Here are two examples:

```
SELECT * FROM t1 WHERE (1,2) = (SELECT column1, column2 FROM t2);
SELECT * FROM t1 WHERE ROW(1,2) = (SELECT column1, column2 FROM t2);
```

The queries here are both **TRUE** if table **t2** has a row where **column1 = 1** and **column2 = 2**.

The expressions **(1, 2)** and **ROW(1, 2)** are sometimes called *row constructors*. The two are equivalent. They are legal in other contexts as well. For example, the following two statements are semantically equivalent (although the first one cannot be optimized until MySQL 5.1.12):

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

The normal use of row constructors is for comparisons with subqueries that return two or more columns. For example, the following query answers the request, “find all rows in table **t1** that also exist in table **t2**”:

```
SELECT column1,column2,column3
FROM t1
WHERE (column1,column2,column3) IN
(SELECT column1,column2,column3 FROM t2);
```

12.2.8.6. EXISTS and NOT EXISTS

If a subquery returns any rows at all, **EXISTS subquery** is **TRUE**, and **NOT EXISTS subquery** is **FALSE**. For example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Traditionally, an **EXISTS** subquery starts with **SELECT ***, but it could begin with **SELECT 5** or **SELECT column1** or anything at all. MySQL ignores the **SELECT** list in such a subquery, so it makes no difference.

For the preceding example, if **t2** contains any rows, even rows with nothing but **NULL** values, the **EXISTS** condition is **TRUE**. This is actually an unlikely example because a **[NOT] EXISTS** subquery almost always contains correlations. Here are some more realistic examples:

- What kind of store is present in one or more cities?

```
SELECT DISTINCT store_type FROM stores
WHERE EXISTS (SELECT * FROM cities_stores
              WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in no cities?

```
SELECT DISTINCT store_type FROM stores
WHERE NOT EXISTS (SELECT * FROM cities_stores
                  WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in all cities?

```
SELECT DISTINCT store_type FROM stores s1
WHERE NOT EXISTS (
  SELECT * FROM cities WHERE NOT EXISTS (
    SELECT * FROM cities_stores
    WHERE cities_stores.city = cities.city
    AND cities_stores.store_type = stores.store_type));
```

The last example is a double-nested **NOT EXISTS** query. That is, it has a **NOT EXISTS** clause within a **NOT EXISTS** clause. Formally, it answers the question “does a city exist with a store that is not in **Stores**”? But it is easier to say that a nested **NOT EXISTS**

answers the question “is x **TRUE** for all y ?”

12.2.8.7. Correlated Subqueries

A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query. For example:

```
SELECT * FROM t1 WHERE column1 = ANY
(SELECT column1 FROM t2 WHERE t2.column2 = t1.column2);
```

Notice that the subquery contains a reference to a column of **t1**, even though the subquery's **FROM** clause does not mention a table **t1**. So, MySQL looks outside the subquery, and finds **t1** in the outer query.

Suppose that table **t1** contains a row where **column1** = 5 and **column2** = 6; meanwhile, table **t2** contains a row where **column1** = 5 and **column2** = 7. The simple expression **... WHERE column1 = ANY (SELECT column1 FROM t2)** would be **TRUE**, but in this example, the **WHERE** clause within the subquery is **FALSE** (because (5, 6) is not equal to (5, 7)), so the subquery as a whole is **FALSE**.

Scoping rule: MySQL evaluates from inside to outside. For example:

```
SELECT column1 FROM t1 AS x
WHERE x.column1 = (SELECT column1 FROM t2 AS x
WHERE x.column1 = (SELECT column1 FROM t3
WHERE x.column2 = t3.column1));
```

In this statement, **x.column2** must be a column in table **t2** because **SELECT column1 FROM t2 AS x ...** renames **t2**. It is not a column in table **t1** because **SELECT column1 FROM t1 ...** is an outer query that is *farther out*.

For subqueries in **HAVING** or **ORDER BY** clauses, MySQL also looks for column names in the outer select list.

For certain cases, a correlated subquery is optimized. For example:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

Otherwise, they are inefficient and likely to be slow. Rewriting the query as a join might improve performance.

Aggregate functions in correlated subqueries may contain outer references, provided the function contains nothing but outer references, and provided the function is not contained in another function or expression.

12.2.8.8. Subqueries in the **FROM** clause

Subqueries are legal in a **SELECT** statement's **FROM** clause. The actual syntax is:

```
SELECT ... FROM (subquery) [AS] name ...
```

The **[AS] name** clause is mandatory, because every table in a **FROM** clause must have a name. Any columns in the *subquery* select list must have unique names. You can find this syntax described elsewhere in this manual, where the term used is “derived tables.”

For the sake of illustration, assume that you have this table:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Here is how to use a subquery in the **FROM** clause, using the example table:

```
INSERT INTO t1 VALUES (1, '1', 1.0);
INSERT INTO t1 VALUES (2, '2', 2.0);
SELECT sb1, sb2, sb3
FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
WHERE sb1 > 1;
```

Result: 2, '2', 4.0.

Here is another example: Suppose that you want to know the average of a set of sums for a grouped table. This does not work:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```


However, this query provides the desired information:

```
SELECT AVG(sum_column1)
FROM (SELECT SUM(column1) AS sum_column1
FROM t1 GROUP BY column1) AS t1;
```

Notice that the column name used within the subquery (`sum_column1`) is recognized in the outer query.

Subqueries in the `FROM` clause can return a scalar, column, row, or table. Subqueries in the `FROM` clause cannot be correlated subqueries, unless used within the `ON` clause of a `JOIN` operation.

Subqueries in the `FROM` clause are executed even for the `EXPLAIN` statement (that is, derived temporary tables are built). This occurs because upper-level queries need information about all tables during the optimization phase, and the table represented by a subquery in the `FROM` clause is unavailable unless the subquery is executed.

12.2.8.9. Subquery Errors

There are some errors that apply only to subqueries. This section describes them.

- Unsupported subquery syntax:

```
ERROR 1235 (ER_NOT_SUPPORTED_YET)
SQLSTATE = 42000
Message = "This version of MySQL does not yet support
'LIMIT & IN/ALL/ANY/SOME subquery'"
```

This means that statements of the following form do not work yet:

```
SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
```

- Incorrect number of columns from subquery:

```
ERROR 1241 (ER_OPERAND_COL)
SQLSTATE = 21000
Message = "Operand should contain 1 column(s)"
```

This error occurs in cases like this:

```
SELECT (SELECT column1, column2 FROM t2) FROM t1;
```

You may use a subquery that returns multiple columns, if the purpose is comparison. In other contexts, the subquery must be a scalar operand. See [Section 12.2.8.5, “Row Subqueries”](#).

- Incorrect number of rows from subquery:

```
ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
SQLSTATE = 21000
Message = "Subquery returns more than 1 row"
```

This error occurs for statements where the subquery returns more than one row. Consider the following example:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

If `SELECT column1 FROM t2` returns just one row, the previous query will work. If the subquery returns more than one row, error 1242 will occur. In that case, the query should be rewritten as:

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- Incorrectly used table in subquery:

```
Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"
```

This error occurs in cases such as the following:

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

You can use a subquery for assignment within an **UPDATE** statement because subqueries are legal in **UPDATE** and **DELETE** statements as well as in **SELECT** statements. However, you cannot use the same table (in this case, table **t1**) for both the subquery's **FROM** clause and the update target.

For transactional storage engines, the failure of a subquery causes the entire statement to fail. For non-transactional storage engines, data modifications made before the error was encountered are preserved.

12.2.8.10. Optimizing Subqueries

Development is ongoing, so no optimization tip is reliable for the long term. The following list provides some interesting tricks that you might want to play with:

- Use subquery clauses that affect the number or order of the rows in the subquery. For example:

```
SELECT * FROM t1 WHERE t1.column1 IN  
(SELECT column1 FROM t2 ORDER BY column1);  
SELECT * FROM t1 WHERE t1.column1 IN  
(SELECT DISTINCT column1 FROM t2);  
SELECT * FROM t1 WHERE EXISTS  
(SELECT * FROM t2 LIMIT 1);
```

- Replace a join with a subquery. For example, try this:

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (  
SELECT column1 FROM t2);
```

Instead of this:

```
SELECT DISTINCT t1.column1 FROM t1, t2  
WHERE t1.column1 = t2.column1;
```

- Some subqueries can be transformed to joins for compatibility with older versions of MySQL that do not support subqueries. However, in some cases, converting a subquery to a join may improve performance. See [Section 12.2.8.11, “Rewriting Subqueries as Joins”](#).
- Move clauses from outside to inside the subquery. For example, use this query:

```
SELECT * FROM t1  
WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

Instead of this query:

```
SELECT * FROM t1  
WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

For another example, use this query:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

Instead of this query:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- Use a row subquery instead of a correlated subquery. For example, use this query:

```
SELECT * FROM t1  
WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

Instead of this query:

```
SELECT * FROM t1
WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
AND t2.column2=t1.column2);
```

- Use `NOT (a = ANY (...))` rather than `a <> ALL (...)`.
- Use `x = ANY (table containing (1,2))` rather than `x=1 OR x=2`.
- Use `= ANY` rather than `EXISTS`.
- For uncorrelated subqueries that always return one row, `IN` is always slower than `=`. For example, use this query:

```
SELECT * FROM t1 WHERE t1.col_name
= (SELECT a FROM t2 WHERE b = some_const);
```

Instead of this query:

```
SELECT * FROM t1 WHERE t1.col_name
IN (SELECT a FROM t2 WHERE b = some_const);
```

These tricks might cause programs to go faster or slower. Using MySQL facilities like the `BENCHMARK()` function, you can get an idea about what helps in your own situation. See [Section 11.11.3, “Information Functions”](#).

Some optimizations that MySQL itself makes are:

- MySQL executes uncorrelated subqueries only once. Use `EXPLAIN` to make sure that a given subquery really is uncorrelated.
- MySQL rewrites `IN`, `ALL`, `ANY`, and `SOME` subqueries in an attempt to take advantage of the possibility that the select-list columns in the subquery are indexed.
- MySQL replaces subqueries of the following form with an index-lookup function, which `EXPLAIN` describes as a special join type (`unique_subquery` or `index_subquery`):

```
... IN (SELECT indexed_column FROM single_table ...)
```

- MySQL enhances expressions of the following form with an expression involving `MIN()` or `MAX()`, unless `NULL` values or empty sets are involved:

```
value {ALL|ANY|SOME} {> | < | >= | <=} (uncorrelated subquery)
```

For example, this `WHERE` clause:

```
WHERE 5 > ALL (SELECT x FROM t)
```

might be treated by the optimizer like this:

```
WHERE 5 > (SELECT MAX(x) FROM t)
```

See also the MySQL Internals Manual chapter [How MySQL Transforms Subqueries](#).

12.2.8.11. Rewriting Subqueries as Joins

Although MySQL 5.1 supports subqueries (see [Section 12.2.8, “Subquery Syntax”](#)), it is still true that there are sometimes other ways to test membership in a set of values. It is also true that on some occasions, it is not only possible to rewrite a query without a subquery, but it can be more efficient to make use of some of these techniques rather than to use subqueries. One of these is the `IN()` construct:

For example, this query:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

Can be rewritten as:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

The queries:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

Can be rewritten using `IN()`:

```
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id
WHERE table2.id IS NULL;
```

A `LEFT [OUTER] JOIN` can be faster than an equivalent subquery because the server might be able to optimize it better — a fact that is not specific to MySQL Server alone. Prior to SQL-92, outer joins did not exist, so subqueries were the only way to do certain things. Today, MySQL Server and many other modern database systems offer a wide range of outer join types.

MySQL Server supports multiple-table `DELETE` statements that can be used to efficiently delete rows based on information from one table or even from many tables at the same time. Multiple-table `UPDATE` statements are also supported. See [Section 12.2.1, “DELETE Syntax”](#), and [Section 12.2.10, “UPDATE Syntax”](#).

12.2.9. TRUNCATE Syntax

```
TRUNCATE [TABLE] tbl_name
```

`TRUNCATE TABLE` empties a table completely. Logically, this is equivalent to a `DELETE` statement that deletes all rows, but there are practical differences under some circumstances.

For `InnoDB` tables, `TRUNCATE TABLE` is mapped to `DELETE` if there are foreign key constraints that reference the table; otherwise fast truncation (dropping and re-creating the table) is used. The `AUTO_INCREMENT` counter is reset by `TRUNCATE TABLE`, regardless of whether there is a foreign key constraint.

For other storage engines, `TRUNCATE TABLE` differs from `DELETE` in the following ways in MySQL 5.1:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one.
- Truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction or active table lock.
- The number of deleted rows is not returned.
- As long as the table format file `tbl_name.frm` is valid, the table can be re-created as an empty table with `TRUNCATE TABLE`, even if the data or index files have become corrupted.
- The table handler does not remember the last used `AUTO_INCREMENT` value, but starts counting from the beginning. This is true even for `MyISAM` and `InnoDB`, which normally do not reuse sequence values.
- When used with partitioned tables, `TRUNCATE TABLE` preserves the partitioning; that is, the data and index files are dropped and re-created, while the partition definitions (`.par`) file is unaffected.
- Since truncation of a table does not make any use of `DELETE`, the `TRUNCATE` statement does not invoke `ON DELETE` triggers.

`TRUNCATE TABLE` requires the `DROP` privilege as of MySQL 5.1.16. (Before 5.1.16, it requires the `DELETE` privilege.)

12.2.10. UPDATE Syntax

Single-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
SET col_name1=expr1 [, col_name2=expr2 ...]
```

```
[WHERE where_condition]  
[ORDER BY ...]  
[LIMIT row_count]
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references  
SET col_name1=expr1 [, col_name2=expr2 ...]  
[WHERE where_condition]
```

For the single-table syntax, the **UPDATE** statement updates columns of existing rows in *tbl_name* with new values. The **SET** clause indicates which columns to modify and the values they should be given. The **WHERE** clause, if given, specifies the conditions that identify which rows to update. With no **WHERE** clause, all rows are updated. If the **ORDER BY** clause is specified, the rows are updated in the order that is specified. The **LIMIT** clause places a limit on the number of rows that can be updated.

For the multiple-table syntax, **UPDATE** updates rows in each table named in *table_references* that satisfy the conditions. In this case, **ORDER BY** and **LIMIT** cannot be used.

where_condition is an expression that evaluates to true for each row to be updated. It is specified as described in [Section 12.2.7, “SELECT Syntax”](#).

The **UPDATE** statement supports the following modifiers:

- If you use the **LOW_PRIORITY** keyword, execution of the **UPDATE** is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (**MyISAM**, **MEMORY**, **MERGE**).
- If you use the **IGNORE** keyword, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur are not updated. Rows for which columns are updated to values that would cause data conversion errors are updated to the closest valid values instead.

If you access a column from *tbl_name* in an expression, **UPDATE** uses the current value of the column. For example, the following statement sets the **age** column to one more than its current value:

```
UPDATE persondata SET age=age+1;
```

Single-table **UPDATE** assignments are generally evaluated from left to right. For multiple-table updates, there is no guarantee that assignments are carried out in any particular order.

If you set a column to the value it currently has, MySQL notices this and does not update it.

If you update a column that has been declared **NOT NULL** by setting to **NULL**, the column is set to the default value appropriate for the data type and the warning count is incremented. The default value is **0** for numeric types, the empty string (**' '**) for string types, and the “zero” value for date and time types.

UPDATE returns the number of rows that were actually changed. The `mysql_info()` C API function returns the number of rows that were matched and updated and the number of warnings that occurred during the **UPDATE**.

You can use **LIMIT row_count** to restrict the scope of the **UPDATE**. A **LIMIT** clause is a rows-matched restriction. The statement stops as soon as it has found *row_count* rows that satisfy the **WHERE** clause, whether or not they actually were changed.

If an **UPDATE** statement includes an **ORDER BY** clause, the rows are updated in the order specified by the clause. This can be useful in certain situations that might otherwise result in an error. Suppose that a table **t** contains a column **id** that has a unique index. The following statement could fail with a duplicate-key error, depending on the order in which rows are updated:

```
UPDATE t SET id = id + 1;
```

For example, if the table contains 1 and 2 in the **id** column and 1 is updated to 2 before 2 is updated to 3, an error occurs. To avoid this problem, add an **ORDER BY** clause to cause the rows with larger **id** values to be updated before those with smaller values:

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

You can also perform **UPDATE** operations covering multiple tables. However, you cannot use **ORDER BY** or **LIMIT** with a multiple-table **UPDATE**. The *table_references* clause lists the tables involved in the join. Its syntax is described in [Section 12.2.7.1, “JOIN](#)

[Syntax](#)". Here is an example:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

The preceding example shows an inner join that uses the comma operator, but multiple-table [UPDATE](#) statements can use any type of join allowed in [SELECT](#) statements, such as [LEFT JOIN](#).

You need the [UPDATE](#) privilege only for columns referenced in a multiple-table [UPDATE](#) that are actually updated. You need only the [SELECT](#) privilege for any columns that are read but not modified.

If you use a multiple-table [UPDATE](#) statement involving [InnoDB](#) tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, update a single table and rely on the [ON UPDATE](#) capabilities that [InnoDB](#) provides to cause the other tables to be modified accordingly. See [Section 13.5.6.4, "FOREIGN KEY Constraints"](#).

Currently, you cannot update a table and select from the same table in a subquery.

12.3. MySQL Utility Statements

12.3.1. DESCRIBE Syntax

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

[DESCRIBE](#) provides information about the columns in a table. It is a shortcut for [SHOW COLUMNS FROM](#). These statements also display information for views. (See [Section 12.5.4.4, "SHOW COLUMNS Syntax"](#).)

[col_name](#) can be a column name, or a string containing the SQL [%](#) and [_](#) wildcard characters to obtain output only for the columns with names matching the string. There is no need to enclose the string within quotes unless it contains spaces or other special characters.

```
mysql> DESCRIBE city;
```

Field	Type	Null	Key	Default	Extra
Id	int(11)	NO	PRI	NULL	auto_increment
Name	char(35)	NO			
Country	char(3)	NO	UNI		
District	char(20)	YES	MUL		
Population	int(11)	NO		0	

5 rows in set (0.00 sec)

[Field](#) indicates the column name.

The [Null](#) field indicates whether [NULL](#) values can be stored in the column.

The [Key](#) field indicates whether the column is indexed. A value of [PRI](#) indicates that the column is part of the table's primary key. [UNI](#) indicates that the column is part of a [UNIQUE](#) index. The [MUL](#) value indicates that multiple occurrences of a given value are allowed within the column.

One reason for [MUL](#) to be displayed on a [UNIQUE](#) index is that several columns form a composite [UNIQUE](#) index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value. Note that in a composite index, only the leftmost column of the index has an entry in the [Key](#) field.

The [Default](#) field indicates the default value that is assigned to the column.

The [Extra](#) field contains any additional information that is available about a given column. In the example shown, the [Extra](#) field indicates that the [Id](#) column was created with the [AUTO_INCREMENT](#) keyword.

If the data types are different from what you expect them to be based on a [CREATE TABLE](#) statement, note that MySQL sometimes changes data types. See [Section 12.1.10.1, "Silent Column Specification Changes"](#).

The [DESCRIBE](#) statement is provided for compatibility with Oracle.

The [SHOW CREATE TABLE](#) and [SHOW TABLE STATUS](#) statements also provide information about tables. See [Section 12.5.4, "SHOW Syntax"](#).

12.3.2. **HELP** Syntax

```
HELP 'search_string'
```

The **HELP** statement returns online information from the MySQL Reference manual. Its proper operation requires that the help tables in the **mysql** database be initialized with help topic information (see [Section 5.2.8, “Server-Side Help”](#)).

The **HELP** statement searches the help tables for the given search string and displays the result of the search. The search string is not case sensitive.

The **HELP** statement understands several types of search strings:

- At the most general level, use **contents** to retrieve a list of the top-level help categories:

```
HELP 'contents'
```

- For a list of topics in a given help category, such as **Data Types**, use the category name:

```
HELP 'data types'
```

- For help on a specific help topic, such as the **ASCII()** function or the **CREATE TABLE** statement, use the associated keyword or keywords:

```
HELP 'ascii'
HELP 'create table'
```

In other words, the search string matches a category, many topics, or a single topic. You cannot necessarily tell in advance whether a given search string will return a list of items or the help information for a single help topic. However, you can tell what kind of response **HELP** returned by examining the number of rows and columns in the result set.

The following descriptions indicate the forms that the result set can take. Output for the example statements is shown using the familiar “tabular” or “vertical” format that you see when using the **mysql** client, but note that **mysql** itself reformats **HELP** result sets in a different way.

- Empty result set

No match could be found for the search string.

- Result set containing a single row with three columns

This means that the search string yielded a hit for the help topic. The result has three columns:

- name**: The topic name.
- description**: Descriptive help text for the topic.
- example**: Usage example or examples. This column might be blank.

Example: **HELP 'replace'**

Yields:

```
name: REPLACE
description: Syntax:
REPLACE(str,from_str,to_str)
```

```
Returns the string str with all occurrences of the string from_str
replaced by the string to_str. REPLACE() performs a case-sensitive
match when searching for from_str.
example: mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'
```

- Result set containing multiple rows with two columns

This means that the search string matched many help topics. The result set indicates the help topic names:

- **name**: The help topic name.
- **is_it_category**: **Y** if the name represents a help category, **N** if it does not. If it does not, the **name** value when specified as the argument to the **HELP** statement should yield a single-row result set containing a description for the named item.

Example: **HELP 'status'**

Yields:

name	is_it_category
SHOW	N
SHOW ENGINE	N
SHOW INNODB STATUS	N
SHOW MASTER STATUS	N
SHOW PROCEDURE STATUS	N
SHOW SLAVE STATUS	N
SHOW SLAVE STATUS	N
SHOW TABLE STATUS	N

- Result set containing multiple rows with three columns

This means the search string matches a category. The result set contains category entries:

- **source_category_name**: The help category name.
- **name**: The category or topic name
- **is_it_category**: **Y** if the name represents a help category, **N** if it does not. If it does not, the **name** value when specified as the argument to the **HELP** statement should yield a single-row result set containing a description for the named item.

Example: **HELP 'functions'**

Yields:

source_category_name	name	is_it_category
Functions	CREATE FUNCTION	N
Functions	DROP FUNCTION	N
Functions	Bit Functions	Y
Functions	Comparison operators	Y
Functions	Control flow functions	Y
Functions	Date and Time Functions	Y
Functions	Encryption Functions	Y
Functions	Information Functions	Y
Functions	Logical operators	Y
Functions	Miscellaneous Functions	Y
Functions	Numeric Functions	Y
Functions	String Functions	Y

Before MySQL 5.1.17, if you intend to use the **HELP()** statement while other tables are locked with **LOCK TABLES**, you must also lock the required **mysql.help_XXX** tables. See [Section 12.4.5, “LOCK TABLES and UNLOCK TABLES Syntax”](#).

12.3.3. USE Syntax

USE db_name

The **USE db_name** statement tells MySQL to use the **db_name** database as the default (current) database for subsequent statements. The database remains the default until the end of the session or another **USE** statement is issued:

```
USE db1;
SELECT COUNT(*) FROM mytable;  # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable;  # selects from db2.mytable
```


Making a particular database the default by means of the `USE` statement does not preclude you from accessing tables in other databases. The following example accesses the `author` table from the `db1` database and the `editor` table from the `db2` database:

```
USE db1;
SELECT author_name, editor_name FROM author, db2.editor
WHERE author.editor_id = db2.editor.editor_id;
```

The `USE` statement is provided for compatibility with Sybase.

12.4. MySQL Transactional and Locking Statements

MySQL supports local transactions (within a given client connection) through statements such as `SET AUTOCOMMIT`, `START TRANSACTION`, `COMMIT`, and `ROLLBACK`. See [Section 12.4.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#). XA transaction support enables MySQL to participate in distributed transactions as well. See [Section 12.4.7, “XA Transactions”](#).

12.4.1. START TRANSACTION, COMMIT, and ROLLBACK Syntax

```
START TRANSACTION [WITH CONSISTENT SNAPSHOT] | BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET AUTOCOMMIT = {0 | 1}
```

The `START TRANSACTION` and `BEGIN` statement begin a new transaction. `COMMIT` commits the current transaction, making its changes permanent. `ROLLBACK` rolls back the current transaction, canceling its changes. The `SET AUTOCOMMIT` statement disables or enables the default autocommit mode for the current connection.

The optional `WORK` keyword is supported for `COMMIT` and `ROLLBACK`, as are the `CHAIN` and `RELEASE` clauses. `CHAIN` and `RELEASE` can be used for additional control over transaction completion. The value of the `completion_type` system variable determines the default completion behavior. See [Section 5.2.3, “System Variables”](#).

The `AND CHAIN` clause causes a new transaction to begin as soon as the current one ends, and the new transaction has the same isolation level as the just-terminated transaction. The `RELEASE` clause causes the server to disconnect the current client connection after terminating the current transaction. Including the `NO` keyword suppresses `CHAIN` or `RELEASE` completion, which can be useful if the `completion_type` system variable is set to cause chaining or release completion by default.

By default, MySQL runs with autocommit mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk.

If you are using a transaction-safe storage engine (such as `InnoDB`, or `NDB Cluster`), you can disable autocommit mode with the following statement:

```
SET AUTOCOMMIT=0;
```

After disabling autocommit mode by setting the `AUTOCOMMIT` variable to zero, you must use `COMMIT` to store your changes to disk or `ROLLBACK` if you want to ignore the changes you have made since the beginning of your transaction.

To disable autocommit mode for a single series of statements, use the `START TRANSACTION` statement:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

With `START TRANSACTION`, autocommit remains disabled until you end the transaction with `COMMIT` or `ROLLBACK`. The autocommit mode then reverts to its previous state.

`BEGIN` and `BEGIN WORK` are supported as aliases of `START TRANSACTION` for initiating a transaction. `START TRANSACTION` is standard SQL syntax and is the recommended way to start an ad-hoc transaction.

Important

Many APIs used for writing MySQL client applications (such as JDBC) provide their own methods for starting transactions that can (and sometimes should) be used instead of sending a `START TRANSACTION` statement from the client. See [Chapter 25, APIs and Libraries](#), or the documentation for your API, for more information.

The **BEGIN** statement differs from the use of the **BEGIN** keyword that starts a **BEGIN . . . END** compound statement. The latter does not begin a transaction. See [Section 19.2.5, “BEGIN . . . END Compound Statement Syntax”](#).

You can also begin a transaction like this:

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

The **WITH CONSISTENT SNAPSHOT** clause starts a consistent read for storage engines that are capable of it. Currently, this applies only to **InnoDB**. The effect is the same as issuing a **START TRANSACTION** followed by a **SELECT** from any **InnoDB** table. See [Section 13.5.10.4, “Consistent Non-Locking Read”](#).

The **WITH CONSISTENT SNAPSHOT** clause does not change the current transaction isolation level, so it provides a consistent snapshot only if the current isolation level is one that allows consistent read (**REPEATABLE READ** or **SERIALIZABLE**).

Beginning a transaction causes any pending transaction to be committed. See [Section 12.4.3, “Statements That Cause an Implicit Commit”](#), for more information.

Beginning a transaction also causes table locks acquired with **LOCK TABLES** to be released, as though you had executed **UNLOCK TABLES**. Beginning a transaction does not release a global read lock acquired with **FLUSH TABLES WITH READ LOCK**.

For best results, transactions should be performed using only tables managed by a single transactional storage engine. Otherwise, the following problems can occur:

- If you use tables from more than one transaction-safe storage engine (such as **InnoDB** and **Falcon**), and the transaction isolation level is not **SERIALIZABLE**, it is possible that when one transaction commits, another ongoing transaction that uses the same tables will see only some of the changes made by the first transaction. That is, the atomicity of transactions is not guaranteed with mixed engines and inconsistencies can result. (If mixed-engine transactions are infrequent, you can use **SET TRANSACTION ISOLATION LEVEL** to set the isolation level to **SERIALIZABLE** on a per-transaction basis as necessary.)
- If you use non-transaction-safe tables within a transaction, any changes to those tables are stored at once, regardless of the status of autocommit mode.

If you issue a **ROLLBACK** statement after updating a non-transactional table within a transaction, an **ER_WARNING_NOT_COMPLETE_ROLLBACK** warning occurs. Changes to transaction-safe tables are rolled back, but not changes to non-transaction-safe tables.

Each transaction is stored in the binary log in one chunk, upon **COMMIT**. Transactions that are rolled back are not logged. (**Exception:** Modifications to non-transactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to non-transactional tables, the entire transaction is logged with a **ROLLBACK** statement at the end to ensure that the modifications to those tables are replicated.) See [Section 5.11.4, “The Binary Log”](#).

You can change the isolation level for transactions with **SET TRANSACTION ISOLATION LEVEL**. See [Section 12.4.6, “SET TRANSACTION Syntax”](#).

Rolling back can be a slow operation that may occur without the user having explicitly asked for it (for example, when an error occurs). Because of this, **SHOW PROCESSLIST** displays **Rolling back** in the **State** column for the connection during implicit and explicit (**ROLLBACK** SQL statement) rollbacks.

12.4.2. Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, those that create, drop, or alter tables or stored routines.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back in such cases by issuing a **ROLLBACK** statement.

12.4.3. Statements That Cause an Implicit Commit

Each of the following statements (and any synonyms for them) implicitly end a transaction, as if you had done a **COMMIT** before executing the statement:

- `ALTER EVENT`, `ALTER FUNCTION`, `ALTER FUNCTION`, `ALTER PROCEDURE`, `ALTER TABLE`, `BEGIN`, `CREATE DATABASE`, `CREATE EVENT`, `CREATE FUNCTION`, `CREATE INDEX`, `CREATE PROCEDURE`, `CREATE TABLE`, `DROP DATABASE`, `DROP EVENT`, `DROP FUNCTION`, `DROP INDEX`, `DROP PROCEDURE`, `DROP TABLE`, `LOAD DATA INFILE`, `LOCK TABLES`, `RENAME TABLE`, `SET AUTOCOMMIT=1`, `START TRANSACTION`, `TRUNCATE TABLE`, `UNLOCK TABLES`.

The `BEGIN` statement differs from the use of the `BEGIN` keyword that starts a `BEGIN . . . END` compound statement. The latter does not cause an implicit commit. See [Section 19.2.5, “BEGIN . . . END Compound Statement Syntax”](#).

- Beginning with MySQL 5.1.3, `ALTER VIEW`, `CREATE TRIGGER`, `CREATE USER`, `CREATE VIEW`, `DROP TRIGGER`, `DROP USER`, `DROP VIEW`, and `RENAME USER` cause an implicit commit.
- `UNLOCK TABLES` commits a transaction only if any tables currently have been locked with `LOCK TABLES`. This does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table-level locks.
- The `CREATE TABLE` statement in `InnoDB` is processed as a single transaction. This means that a `ROLLBACK` from the user does not undo `CREATE TABLE` statements the user made during that transaction.
- `CREATE TABLE` and `DROP TABLE` do not commit a transaction if the `TEMPORARY` keyword is used. (This does not apply to other operations on temporary tables such as `CREATE INDEX`, which do cause a commit.)
- In MySQL 5.1.11 and earlier, `LOAD DATA INFILE` caused an implicit commit for all storage engines. Beginning with MySQL 5.1.12, it causes an implicit commit only for tables using the `NDB` storage engine. For more information, see [Bug#11151](#).
- Beginning with MySQL 5.1.15, `CREATE TABLE . . . SELECT` causes an implicit commit before and after the statement is executed when you are creating non-temporary tables. (No commit occurs for `CREATE TEMPORARY TABLE . . . SELECT`.) This is to prevent an issue during replication where the table could be created on the master after a rollback, but fail to be recorded in the binary log, and therefore not replicated to the slave. For more information, see [Bug#22865](#).

Transactions cannot be nested. This is a consequence of the implicit `COMMIT` performed for any current transaction when you issue a `START TRANSACTION` statement or one of its synonyms.

Statements that cause implicit cannot be used in an XA transaction while the transaction is in an `ACTIVE` state.

12.4.4. SAVEPOINT and ROLLBACK TO SAVEPOINT Syntax

```
SAVEPOINT identifier
ROLLBACK [WORK] TO SAVEPOINT identifier
RELEASE SAVEPOINT identifier
```

`InnoDB` supports the SQL statements `SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, `RELEASE SAVEPOINT` and the optional `WORK` keyword for `ROLLBACK`.

The `SAVEPOINT` statement sets a named transaction savepoint with a name of *identifier*. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The `ROLLBACK TO SAVEPOINT` statement rolls back a transaction to the named savepoint. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but `InnoDB` does *not* release the row locks that were stored in memory after the savepoint. (Note that for a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

If the `ROLLBACK TO SAVEPOINT` statement returns the following error, it means that no savepoint with the specified name exists:

```
ERROR 1181: Got error 153 during ROLLBACK
```

The `RELEASE SAVEPOINT` statement removes the named savepoint from the set of savepoints of the current transaction. No commit or rollback occurs. It is an error if the savepoint does not exist.

All savepoints of the current transaction are deleted if you execute a `COMMIT`, or a `ROLLBACK` that does not name a savepoint.

Beginning with MySQL 5.0.17, a new savepoint level is created when a stored function is invoked or a trigger is activated. The savepoints on previous levels become unavailable and thus do not conflict with savepoints on the new level. When the function or trigger terminates, any savepoints it created are released and the previous savepoint level is restored.

12.4.5. LOCK TABLES and UNLOCK TABLES Syntax

```
LOCK TABLES
    tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type] ...

lock_type:
    READ [LOCAL]
    | [LOW_PRIORITY] WRITE

UNLOCK TABLES
```

LOCK TABLES locks base tables (but not views) for the current thread. To use **LOCK TABLES**, you must have the **LOCK TABLES** privilege, and the **SELECT** privilege for each table to be locked.

UNLOCK TABLES explicitly releases any table locks held by the current thread. Another use for **UNLOCK TABLES** is to release the global read lock acquired with **FLUSH TABLES WITH READ LOCK**. (You can lock all tables in all databases with read locks with the **FLUSH TABLES WITH READ LOCK** statement. See [Section 12.5.5.2, “FLUSH Syntax”](#). This is a very convenient way to get backups if you have a filesystem such as Veritas that can take snapshots in time.)

The following general rules apply to acquisition and release of locks by a given thread:

- Table locks are acquired with **LOCK TABLES**.
- If the **LOCK TABLES** statement must wait due to locks held by other threads on any of the tables, it blocks until all locks can be acquired.
- Table locks are released explicitly with **UNLOCK TABLES**.
- Table locks are released implicitly under these conditions:
 - **LOCK TABLES** releases any table locks currently held by the thread before acquiring new locks.
 - Beginning a transaction (for example, with **START TRANSACTION**) implicitly performs an **UNLOCK TABLES**. (Additional information about the interaction between table locking and transactions is given later in the section.)
 - If a client connection drops, the server releases table locks held by the client. If the client reconnects, the locks will no longer be in effect. For this reason, clients may wish to disable auto-reconnect. With auto-reconnect in effect, the client is not notified if reconnect occurs but any table locks will have been lost. With auto-reconnect disabled, if the connection drops, an error occurs for the next statement issued. The client can detect the error and take appropriate action such as reacquiring the locks. See [Section 25.2.13, “Controlling Automatic Reconnect Behavior”](#).
- One thread cannot release another thread's locks.

Note

If you use **ALTER TABLE** on a locked table, it may become unlocked. See [Section B.1.7.1, “Problems with ALTER TABLE”](#).

The main reasons to use **LOCK TABLES** are to emulate transactions or to get more speed when updating tables. This is explained in more detail later in this section.

A table lock protects only against inappropriate reads or writes by other clients. The client holding the lock, even a read lock, can perform table-level operations such as **DROP TABLE**. Truncate operations are not transaction-safe, so an error occurs if the client attempts one during an active transaction or while holding a table lock.

When you use **LOCK TABLES**, you must lock all tables that you are going to use in your statements. While the locks obtained with a **LOCK TABLES** statement are in effect, you cannot access any tables that were not locked by the statement. Because **LOCK TABLES** will not lock views, if the operation that you are performing uses any views, you must also lock all base tables on which those views depend.

You cannot use a locked table multiple times in a single query. Use aliases instead, in which case you must obtain a lock for each alias separately:

```
mysql> LOCK TABLE t WRITE, t AS t1 WRITE;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
```

```
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

If your statements refer to a table by means of an alias, you must lock the table using that same alias. It does not work to lock the table without specifying the alias:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

Conversely, if you lock a table using an alias, you must refer to it in your statements using that alias:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

If a thread obtains a **READ** lock on a table, that thread (and all other threads) can only read from the table. If a thread obtains a **WRITE** lock on a table, only the thread holding the lock can write to the table. Other threads are blocked from reading or writing the table until the lock has been released.

The difference between **READ LOCAL** and **READ** is that **READ LOCAL** allows non-conflicting **INSERT** statements (concurrent inserts) to execute while the lock is held. However, **READ LOCAL** cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock. For **InnoDB** tables, **READ LOCAL** is the same as **READ**.

WRITE locks normally have higher priority than **READ** locks to ensure that updates are processed as soon as possible. This means that if one thread obtains a **READ** lock and then another thread requests a **WRITE** lock, subsequent **READ** lock requests wait until the **WRITE** thread has gotten the lock and released it. You can use **LOW_PRIORITY WRITE** locks to allow other threads to obtain **READ** locks before the thread that is waiting for the **WRITE** lock. You should use **LOW_PRIORITY WRITE** locks only if you are sure that eventually there will be a time when no threads have a **READ** lock. (Exception: For **InnoDB** tables in transactional mode (autocommit = 0), a **LOW_PRIORITY WRITE** lock acts like a regular **WRITE** lock and precedes waiting **READ** locks.)

LOCK TABLES works as follows:

1. Sort all tables to be locked in an internally defined order. From the user standpoint, this order is undefined.
2. If a table is to be locked with a read and a write lock, put the write lock request before the read lock request.
3. Lock one table at a time until the thread gets all locks.

This policy ensures that table locking is deadlock free. There are, however, other things you need to be aware of about this policy: If you are using a **LOW_PRIORITY WRITE** lock for a table, it means only that MySQL waits for this particular lock until there are no threads that want a **READ** lock. When the thread has gotten the **WRITE** lock and is waiting to get the lock for the next table in the lock table list, all other threads wait for the **WRITE** lock to be released. If this becomes a serious problem with your application, you should consider converting some of your tables to transaction-safe tables.

For some operations, system tables in the **mysql** database must be accessed. For example, the **HELP** statement requires the contents of the server-side help tables, and **CONVERT_TZ()** might need to read the time zone tables. Before MySQL 5.1.17, to perform such operations while a **LOCK TABLES** statement is in effect, you must also lock the requisite system tables explicitly or a lock error occurs. As of 5.1.17, the server implicitly locks the system tables for reading as necessary so that you need not lock them explicitly. These tables are treated as just described:

```
mysql.help_category
mysql.help_keyword
mysql.help_relation
mysql.help_topic
mysql.proc
mysql.time_zone
mysql.time_zone_leap_second
mysql.time_zone_name
mysql.time_zone_transition
mysql.time_zone_transition_type
```

If you want to explicitly place a **WRITE** lock on any of those tables with a **LOCK TABLES** statement, the table must be the only one locked; no other table can be locked with the same statement.

LOCK TABLES and **UNLOCK TABLES** interact with the use of transactional tables as follows:

- **LOCK TABLES** is not transaction-safe and implicitly commits any active transaction before attempting to lock the tables. Also, beginning a transaction (for example, with **START TRANSACTION**) implicitly performs an **UNLOCK TABLES**. (See [Section 12.4.3, “Statements That Cause an Implicit Commit”](#).)
- **UNLOCK TABLES** implicitly commits any active transaction, but only if any tables have been locked with **LOCK TABLES**.
- The correct way to use **LOCK TABLES** and **UNLOCK TABLES** with transactional tables, such as **InnoDB** tables, is to set **AUTO-COMMIT = 0** and not to call **UNLOCK TABLES** until you commit the transaction explicitly. When you call **LOCK TABLES**, **InnoDB** internally takes its own table lock, and MySQL takes its own table lock. **InnoDB** releases its table lock at the next commit, but for MySQL to release its table lock, you have to call **UNLOCK TABLES**. You should not have **AUTOCOMMIT = 1**, because then **InnoDB** releases its table lock immediately after the call of **LOCK TABLES**, and deadlocks can very easily happen. Note that we do not acquire the **InnoDB** table lock at all if **AUTOCOMMIT=1**, to help old applications avoid unnecessary deadlocks.
- **ROLLBACK** does not release MySQL's non-transactional table locks.
- **FLUSH TABLES WITH READ LOCK** acquires a global read lock and not table locks, so it is not subject to the same behavior as **LOCK TABLES** and **UNLOCK TABLES** with respect to table locking and implicit commits. See [Section 12.5.5.2, “FLUSH Syntax”](#).

You can safely use **KILL** to terminate a thread that is waiting for a table lock. See [Section 12.5.5.3, “KILL Syntax”](#).

Note that you should *not* lock any tables that you are using with **INSERT DELAYED** because in that case the **INSERT** is performed by a separate thread.

Normally, you do not need to lock tables, because all single **UPDATE** statements are atomic; no other thread can interfere with any other currently executing SQL statement. However, there are a few cases when locking tables may provide an advantage:

- If you are going to run many operations on a set of **MyISAM** tables, it is much faster to lock the tables you are going to use. Locking **MyISAM** tables speeds up inserting, updating, or deleting on them because MySQL does not flush the key cache for the locked tables until **UNLOCK TABLES** is called. Normally, the key cache is flushed after each SQL statement.

The downside to locking the tables is that no thread can update a **READ**-locked table (including the one holding the lock) and no thread can access a **WRITE**-locked table other than the one holding the lock.

- If you are using tables for a non-transactional storage engine, you must use **LOCK TABLES** if you want to ensure that no other thread modifies the tables between a **SELECT** and an **UPDATE**. The example shown here requires **LOCK TABLES** to execute safely:

```
LOCK TABLES trans READ, customer WRITE;
SELECT SUM(value) FROM trans WHERE customer_id=some_id;
UPDATE customer
  SET total_value=sum_from_previous_statement
  WHERE customer_id=some_id;
UNLOCK TABLES;
```

Without **LOCK TABLES**, it is possible that another thread might insert a new row in the **trans** table between execution of the **SELECT** and **UPDATE** statements.

You can avoid using **LOCK TABLES** in many cases by using relative updates (**UPDATE customer SET value=value+new_value**) or the **LAST_INSERT_ID()** function. See [Section 1.9.5.2, “Transactions and Atomic Operations”](#).

You can also avoid locking tables in some cases by using the user-level advisory lock functions **GET_LOCK()** and **RELEASE_LOCK()**. These locks are saved in a hash table in the server and implemented with **pthread_mutex_lock()** and **pthread_mutex_unlock()** for high speed. See [Section 11.11.4, “Miscellaneous Functions”](#).

See [Section 6.3.1, “Internal Locking Methods”](#), for more information on locking policy.

12.4.6. SET TRANSACTION Syntax

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

This statement sets the transaction isolation level for the next transaction, globally, or for the current session.

The default behavior of `SET TRANSACTION` is to set the isolation level for the next (not yet started) transaction. If you use the `GLOBAL` keyword, the statement sets the default transaction level globally for all new connections created from that point on. Existing connections are unaffected. You need the `SUPER` privilege to do this. Using the `SESSION` keyword sets the default transaction level for all future transactions performed on the current connection.

For descriptions of each `InnoDB` transaction isolation level, see [Section 13.5.10.3, “InnoDB and TRANSACTION ISOLATION LEVEL”](#). `InnoDB` supports each of these levels in MySQL 5.1. The default level is `REPEATABLE READ`.

In MySQL 5.1, if the `READ COMMITTED` isolation level is used or the `innodb_locks_unsafe_for_binlog` system variable is enabled, there is no `InnoDB` gap locking except in constraint checking. Also, record locks for non-matching rows are released after MySQL has evaluated the `WHERE` condition.

To set the initial default global isolation level for `mysqld`, use the `--transaction-isolation` option. See [Section 5.2.2, “Command Options”](#).

12.4.7. XA Transactions

Support for XA transactions is available for the `InnoDB` storage engine. The MySQL XA implementation is based on the X/Open CAE document *Distributed Transaction Processing: The XA Specification*. This document is published by The Open Group and available at <http://www.opengroup.org/public/pubs/catalog/c193.htm>. Limitations of the current XA implementation are described in [Section D.5, “Restrictions on XA Transactions”](#).

On the client side, there are no special requirements. The XA interface to a MySQL server consists of SQL statements that begin with the `XA` keyword. MySQL client programs must be able to send SQL statements and to understand the semantics of the XA statement interface. They do not need be linked against a recent client library. Older client libraries also will work.

Currently, among the MySQL Connectors, MySQL Connector/J 5.0.0 supports XA directly (by means of a class interface that handles the XA SQL statement interface for you).

XA supports distributed transactions; that is, the ability to allow multiple separate transactional resources to participate in a global transaction. Transactional resources often are RDBMSs but may be other kinds of resources.

A global transaction involves several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends ACID properties “up a level” so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties. (However, for a distributed transaction, you must use the `SERIALIZABLE` isolation level to achieve ACID properties. It is enough to use `REPEATABLE READ` for a non-distributed transaction, but not for a distributed transaction.)

Some examples of distributed transactions:

- An application may act as an integration tool that combines a messaging service with an RDBMS. The application makes sure that transactions dealing with message sending, retrieval, and processing that also involve a transactional database all happen in a global transaction. You can think of this as “transactional email.”
- An application performs actions that involve different database servers, such as a MySQL server and an Oracle server (or multiple MySQL servers), where actions that involve multiple servers must happen as part of a global transaction, rather than as separate transactions local to each server.
- A bank keeps account information in an RDBMS and distributes and receives money via automated teller machines (ATMs). It is necessary to ensure that ATM actions are correctly reflected in the accounts, but this cannot be done with the RDBMS alone. A global transaction manager integrates the ATM and database resources to ensure overall consistency of financial transactions.

Applications that use global transactions involve one or more Resource Managers and a Transaction Manager:

- A Resource Manager (RM) provides access to transactional resources. A database server is one kind of resource manager. It must be possible to either commit or roll back transactions managed by the RM.
- A Transaction Manager (TM) coordinates the transactions that are part of a global transaction. It communicates with the RMs that handle each of these transactions. The individual transactions within a global transaction are “branches” of the global transaction. Global transactions and their branches are identified by a naming scheme described later.

The MySQL implementation of XA MySQL enables a MySQL server to act as a Resource Manager that handles XA transactions within a global transaction. A client program that connects to the MySQL server acts as the Transaction Manager.

To carry out a global transaction, it is necessary to know which components are involved, and bring each component to a point when it can be committed or rolled back. Depending on what each component reports about its ability to succeed, they must all commit or roll back as an atomic group. That is, either all components must commit, or all components must roll back. To manage a global transaction, it is necessary to take into account that any component or the connecting network might fail.

The process for executing a global transaction uses two-phase commit (2PC). This takes place after the actions performed by the branches of the global transaction have been executed.

1. In the first phase, all branches are prepared. That is, they are told by the TM to get ready to commit. Typically, this means each RM that manages a branch records the actions for the branch in stable storage. The branches indicate whether they are able to do this, and these results are used for the second phase.
2. In the second phase, the TM tells the RMs whether to commit or roll back. If all branches indicated when they were prepared that they will be able to commit, all branches are told to commit. If any branch indicated when it was prepared that it will not be able to commit, all branches are told to roll back.

In some cases, a global transaction might use one-phase commit (1PC). For example, when a Transaction Manager finds that a global transaction consists of only one transactional resource (that is, a single branch), that resource can be told to prepare and commit at the same time.

12.4.7.1. XA Transaction SQL Syntax

To perform XA transactions in MySQL, use the following statements:

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER
```

For **XA START**, the **JOIN** and **RESUME** clauses are not supported.

For **XA END** the **SUSPEND [FOR MIGRATE]** clause is not supported.

Each XA statement begins with the **XA** keyword, and most of them require an *xid* value. An *xid* is an XA transaction identifier. It indicates which transaction the statement applies to. *xid* values are supplied by the client, or generated by the MySQL server. An *xid* value has from one to three parts:

```
xid: gtrid [, bqual [, formatID ]]
```

gtrid is a global transaction identifier, *bqual* is a branch qualifier, and *formatID* is a number that identifies the format used by the *gtrid* and *bqual* values. As indicated by the syntax, *bqual* and *formatID* are optional. The default *bqual* value is `' '` if not given. The default *formatID* value is 1 if not given.

gtrid and *bqual* must be string literals, each up to 64 bytes (not characters) long. *gtrid* and *bqual* can be specified in several ways. You can use a quoted string (`'ab'`), hex string (`0x6162`, `X'ab'`), or bit value (`b'nnnn'`).

formatID is an unsigned integer.

The *gtrid* and *bqual* values are interpreted in bytes by the MySQL server's underlying XA support routines. However, while an SQL statement containing an XA statement is being parsed, the server works with some specific character set. To be safe, write *gtrid* and *bqual* as hex strings.

xid values typically are generated by the Transaction Manager. Values generated by one TM must be different from values generated by other TMs. A given TM must be able to recognize its own *xid* values in a list of values returned by the **XA RECOVER** statement.

MySQL Enterprise

For expert advice on XA Distributed Transaction Support subscribe to the MySQL Enterprise Monitor. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

XA START *xid* starts an XA transaction with the given *xid* value. Each XA transaction must have a unique *xid* value, so the value must not currently be used by another XA transaction. Uniqueness is assessed using the *gtrid* and *bqual* values. All following XA statements for the XA transaction must be specified using the same *xid* value as that given in the **XA START** statement. If you use any of those statements but specify an *xid* value that does not correspond to some existing XA transaction, an error occurs.

One or more XA transactions can be part of the same global transaction. All XA transactions within a given global transaction must use the same *gtrid* value in the *xid* value. For this reason, *gtrid* values must be globally unique so that there is no ambiguity about which global transaction a given XA transaction is part of. The *bqual* part of the *xid* value must be different for each XA transaction within a global transaction. (The requirement that *bqual* values be different is a limitation of the current MySQL XA implementation. It is not part of the XA specification.)

The **XA RECOVER** statement returns information for those XA transactions on the MySQL server that are in the **PREPARED** state. (See [Section 12.4.7.2, “XA Transaction States”](#).) The output includes a row for each such XA transaction on the server, regardless of which client started it.

XA RECOVER output rows look like this (for an example *xid* value consisting of the parts 'abc', 'def', and 7):

```
mysql> XA RECOVER;
+-----+-----+-----+-----+
| formatID | gtrid_length | bqual_length | data |
+-----+-----+-----+-----+
| 7 | 3 | 3 | abcdef |
+-----+-----+-----+-----+
```

The output columns have the following meanings:

- *formatID* is the *formatID* part of the transaction *xid*
- *gtrid_length* is the length in bytes of the *gtrid* part of the *xid*
- *bqual_length* is the length in bytes of the *bqual* part of the *xid*
- *data* is the concatenation of the *gtrid* and *bqual* parts of the *xid*

12.4.7.2. XA Transaction States

An XA transaction progresses through the following states:

1. Use **XA START** to start an XA transaction and put it in the **ACTIVE** state.
2. For an **ACTIVE** XA transaction, issue the SQL statements that make up the transaction, and then issue an **XA END** statement. **XA END** puts the transaction in the **IDLE** state.
3. For an **IDLE** XA transaction, you can issue either an **XA PREPARE** statement or an **XA COMMIT ... ONE PHASE** statement:
 - **XA PREPARE** puts the transaction in the **PREPARED** state. An **XA RECOVER** statement at this point will include the transaction's *xid* value in its output, because **XA RECOVER** lists all XA transactions that are in the **PREPARED** state.
 - **XA COMMIT ... ONE PHASE** prepares and commits the transaction. The *xid* value will not be listed by **XA RECOVER** because the transaction terminates.
4. For a **PREPARED** XA transaction, you can issue an **XA COMMIT** statement to commit and terminate the transaction, or **XA ROLLBACK** to roll back and terminate the transaction.

Here is a simple XA transaction that inserts a row into a table as part of a global transaction:

```
mysql> XA START 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mytable (i) VALUES(10);
```

```
Query OK, 1 row affected (0.04 sec)

mysql> XA END 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA PREPARE 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA COMMIT 'xatest';
Query OK, 0 rows affected (0.00 sec)
```

Within the context of a given client connection, XA transactions and local (non-XA) transactions are mutually exclusive. For example, if **XA START** has been issued to begin an XA transaction, a local transaction cannot be started until the XA transaction has been committed or rolled back. Conversely, if a local transaction has been started with **START TRANSACTION**, no XA statements can be used until the transaction has been committed or rolled back.

Note that if an XA transaction is in the **ACTIVE** state, you cannot issue any statements that cause an implicit commit. That would violate the XA contract because you could not roll back the XA transaction. You will receive the following error if you try to execute such a statement:

```
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed
when global transaction is in the ACTIVE state
```

Statements to which the preceding remark applies are listed at [Section 12.4.3, “Statements That Cause an Implicit Commit”](#).

12.5. Database Administration Statements

12.5.1. Account Management Statements

MySQL account information is stored in the tables of the `mysql` database. This database and the access control system are discussed extensively in [Chapter 5, Database Administration](#), which you should consult for additional details.

Important

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

MySQL Enterprise

In a production environment it is always prudent to examine any changes to users' accounts. The MySQL Enterprise Monitor provides notification whenever users' privileges are altered. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

12.5.1.1. CREATE USER Syntax

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']
[, user [IDENTIFIED BY [PASSWORD] 'password']] ...
```

The **CREATE USER** statement creates new MySQL accounts. To use it, you must have the global **CREATE USER** privilege or the **INSERT** privilege for the `mysql` database. For each account, **CREATE USER** creates a new row in the `mysql.user` table that has no privileges. An error occurs if the account already exists. Each account is named using the same format as for the **GRANT** statement; for example, `'jeffrey'@'localhost'`. If you specify only the username part of the account name, a hostname part of `'%'` is used. For additional information about specifying account names, see [Section 12.5.1.3, “GRANT Syntax”](#).

The account can be given a password with the optional **IDENTIFIED BY** clause. The *user* value and the password are given the same way as for the **GRANT** statement. In particular, to specify the password in plain text, omit the **PASSWORD** keyword. To specify the password as the hashed value as returned by the **PASSWORD()** function, include the **PASSWORD** keyword. See [Section 12.5.1.3, “GRANT Syntax”](#).

12.5.1.2. DROP USER Syntax

```
DROP USER user [, user] ...
```

The **DROP USER** statement removes one or more MySQL accounts. It removes privilege rows for the account from all grant tables. To

use this statement, you must have the global **CREATE USER** privilege or the **DELETE** privilege for the `mysql` database. Each account is named using the same format as for the **GRANT** statement; for example, `'jeffrey'@'localhost'`. If you specify only the user-name part of the account name, a hostname part of `'%'` is used. For additional information about specifying account names, see [Section 12.5.1.3, “GRANT Syntax”](#).

With **DROP USER**, you can remove an account and its privileges as follows:

```
DROP USER user;
```

Important

DROP USER does not automatically close any open user sessions. Rather, in the event that a user with an open session is dropped, the statement does not take effect until that user's session is closed. Once the session is closed, the user is dropped, and that user's next attempt to log in will fail. *This is by design.*

DROP USER does not automatically delete or invalidate any database objects that the user created. This applies to tables, views, stored routines, triggers, and events.

12.5.1.3. GRANT Syntax

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
ON [object_type] {tbl_name | * | *.* | db_name.*}
TO user [IDENTIFIED BY [PASSWORD] 'password']
    [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
[REQUIRE
    NONE |
    [{SSL | X509}]
    [CIPHER 'cipher' [AND]]
    [ISSUER 'issuer' [AND]]
    [SUBJECT 'subject']]
[WITH with_option [with_option] ...]

object_type =
TABLE
|
FUNCTION
|
PROCEDURE

with_option =
GRANT OPTION
|
MAX_QUERIES_PER_HOUR count
|
MAX_UPDATES_PER_HOUR count
|
MAX_CONNECTIONS_PER_HOUR count
|
MAX_USER_CONNECTIONS count
```

The **GRANT** statement enables system administrators to create MySQL user accounts and to grant rights to from accounts. To use **GRANT**, you must have the **GRANT OPTION** privilege, and you must have the privileges that you are granting. The **REVOKE** statement is related and enables administrators to remove account privileges. See [Section 12.5.1.5, “REVOKE Syntax”](#).

MySQL Enterprise

For automated notification of users with inappropriate privileges, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

MySQL account information is stored in the tables of the `mysql` database. This database and the access control system are discussed extensively in [Chapter 5, Database Administration](#), which you should consult for additional details.

Important

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

If the grant tables hold privilege rows that contain mixed-case database or table names and the `lower_case_table_names` system variable is set to a non-zero value, **REVOKE** cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (**GRANT** will not create such rows when `lower_case_table_names` is set, but such rows might have been created prior to setting the variable.)

Privileges can be granted at several levels. The examples shown here include no **IDENTIFIED BY 'password'** clause for brevity, but you should include one if the account does not already exist to avoid creating an account with no password.

- **Global level**

Global privileges apply to all databases on a given server. These privileges are stored in the `mysql.user` table. `GRANT ALL ON *.*` and `REVOKE ALL ON *.*` grant and revoke only global privileges.

```
GRANT ALL ON *.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON *.* TO 'someuser'@'somehost';
```

- **Database level**

Database privileges apply to all objects in a given database. These privileges are stored in the `mysql.db` and `mysql.host` tables. `GRANT ALL ON db_name.*` and `REVOKE ALL ON db_name.*` grant and revoke only database privileges.

```
GRANT ALL ON mydb.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.* TO 'someuser'@'somehost';
```

- **Table level**

Table privileges apply to all columns in a given table. These privileges are stored in the `mysql.tables_priv` table. `GRANT ALL ON db_name.tbl_name` and `REVOKE ALL ON db_name.tbl_name` grant and revoke only table privileges.

```
GRANT ALL ON mydb.mytbl TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.mytbl TO 'someuser'@'somehost';
```

- **Column level**

Column privileges apply to single columns in a given table. These privileges are stored in the `mysql.columns_priv` table. When using `REVOKE`, you must specify the same columns that were granted. The column or columns for which the privileges are to be granted must be enclosed within parentheses.

```
GRANT SELECT (col1), INSERT (col1,col2) ON mydb.mytbl TO 'someuser'@'somehost';
```

- **Routine level**

The `CREATE ROUTINE`, `ALTER ROUTINE`, `EXECUTE`, and `GRANT` privileges apply to stored routines (functions and procedures). They can be granted at the global and database levels. Also, except for `CREATE ROUTINE`, these privileges can be granted at the routine level for individual routines and are stored in the `mysql.procs_priv` table.

```
GRANT CREATE ROUTINE ON mydb.* TO 'someuser'@'somehost';
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'someuser'@'somehost';
```

The *object_type* clause should be specified as `TABLE`, `FUNCTION`, or `PROCEDURE` when the following object is a table, a stored function, or a stored procedure.

For the `GRANT` and `REVOKE` statements, *priv_type* can be specified as any of the following:

Privilege	Meaning
<code>ALL [PRIVILEGES]</code>	Sets all simple privileges except <code>GRANT OPTION</code>
<code>ALTER</code>	Enables use of <code>ALTER TABLE</code>
<code>ALTER ROUTINE</code>	Enables stored routines to be altered or dropped
<code>CREATE</code>	Enables use of <code>CREATE TABLE</code>
<code>CREATE ROUTINE</code>	Enables creation of stored routines
<code>CREATE TEMPORARY TABLES</code>	Enables use of <code>CREATE TEMPORARY TABLE</code>
<code>CREATE USER</code>	Enables use of <code>CREATE USER</code> , <code>DROP USER</code> , <code>RENAME USER</code> , and <code>REVOKE ALL PRIVILEGES</code> .
<code>CREATE VIEW</code>	Enables use of <code>CREATE VIEW</code>
<code>DELETE</code>	Enables use of <code>DELETE</code>
<code>DROP</code>	Enables use of <code>DROP TABLE</code>
<code>EVENT</code>	Enables creation of events for the event scheduler

EXECUTE	Enables the user to run stored routines
FILE	Enables use of <code>SELECT ... INTO OUTFILE</code> and <code>LOAD DATA INFILE</code>
INDEX	Enables use of <code>CREATE INDEX</code> and <code>DROP INDEX</code>
INSERT	Enables use of <code>INSERT</code>
LOCK TABLES	Enables use of <code>LOCK TABLES</code> on tables for which you have the <code>SELECT</code> privilege
PROCESS	Enables the user to see all processes with <code>SHOW PROCESSLIST</code>
REFERENCES	Not implemented
RELOAD	Enables use of <code>FLUSH</code>
REPLICATION CLIENT	Enables the user to ask where slave or master servers are
REPLICATION SLAVE	Needed for replication slaves (to read binary log events from the master)
SELECT	Enables use of <code>SELECT</code>
SHOW DATABASES	<code>SHOW DATABASES</code> shows all databases
SHOW VIEW	Enables use of <code>SHOW CREATE VIEW</code>
SHUTDOWN	Enables use of <code>mysqladmin shutdown</code>
SUPER	Enables use of <code>CHANGE MASTER</code> , <code>KILL</code> , <code>PURGE MASTER LOGS</code> , and <code>SET GLOBAL</code> statements, the <code>mysqladmin debug</code> command; allows you to connect (once) even if <code>max_connections</code> is reached
TRIGGER	Enables the user to create or drop triggers
UPDATE	Enables use of <code>UPDATE</code>
USAGE	Synonym for “no privileges”
GRANT OPTION	Enables privileges to be granted

The `EVENT` and `TRIGGER` privileges were added in MySQL 5.1.6. A trigger is associated with a table, so to create or drop a trigger, you must have the `TRIGGER` privilege for the table, not the trigger. (Before MySQL 5.1.6, the `SUPER` privilege was required to create or drop triggers.)

The `REFERENCES` privilege currently is unused.

`USAGE` can be specified when you want to create a user that has no privileges.

Use `SHOW GRANTS` to determine what privileges an account has. See [Section 12.5.4.17, “SHOW GRANTS Syntax”](#).

You can assign global privileges by using `ON *.*` syntax or database-level privileges by using `ON db_name.*` syntax. If you specify `ON *` and you have selected a default database, the privileges are granted in that database.

Warning

If you specify `ON *` and you have *not* selected a default database, the privileges granted are global.

The `FILE`, `PROCESS`, `RELOAD`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, `SHOW DATABASES`, `SHUTDOWN`, `SUPER`, and `CREATE USER` privileges are administrative privileges that can only be granted globally (using `ON *.*` syntax).

Other privileges can be granted globally or at more specific levels.

The *priv_type* values that you can specify for a table are `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `DROP`, `GRANT OPTION`, `INDEX`, `ALTER`, `CREATE VIEW`, `SHOW VIEW` and `TRIGGER`.

The *priv_type* values that you can specify for a column (that is, when you use a *column_list* clause) are `SELECT`, `INSERT`, and `UPDATE`.

The *priv_type* values that you can specify at the routine level are `ALTER ROUTINE`, `EXECUTE`, and `GRANT OPTION`. `CREATE ROUTINE` is not a routine-level privilege because you must have this privilege to create a routine in the first place.

For the global, database, table, and routine levels, `GRANT ALL` assigns only the privileges that exist at the level you are granting. For example, `GRANT ALL ON db_name.*` is a database-level statement, so it does not grant any global-only privileges such as `FILE`.

MySQL allows you to grant privileges even on database objects that do not exist. In such cases, the privileges to be granted must include the [CREATE](#) privilege. *This behavior is by design*, and is intended to enable the database administrator to prepare user accounts and privileges for database objects that are to be created at a later time.

Important

MySQL does not automatically revoke any privileges when you drop a table or database. However, if you drop a routine, any routine-level privileges granted for that routine are revoked.

Note

The `'_'` and `'%'` wildcards are allowed when specifying database names in [GRANT](#) statements that grant privileges at the global or database levels. This means, for example, that if you want to use a `'_'` character as part of a database name, you should specify it as `'\_'` in the [GRANT](#) statement, to prevent the user from being able to access additional databases matching the wildcard pattern; for example, [GRANT ... ON `foo\\\_bar`.* TO ...](#).

To accommodate granting rights to users from arbitrary hosts, MySQL supports specifying the *user* value in the form *user_name@host_name*. If a *user_name* or *host_name* value is legal as an unquoted identifier, you need not quote it. However, quotes are necessary to specify a *user_name* string containing special characters (such as `'-'`), or a *host_name* string containing special characters or wildcard characters (such as `'%'`); for example, `'test-user'@'test-hostname'`. Quote the username and hostname separately.

You can specify wildcards in the hostname. For example, *user_name@'%.loc.gov'* applies to *user_name* for any host in the `loc.gov` domain, and *user_name@'144.155.166.%'* applies to *user_name* for any host in the `144.155.166` class C subnet.

The simple form *user_name* is a synonym for *user_name@'%'*.

MySQL does not support wildcards in usernames. Anonymous users are defined by inserting entries with `User=''` into the `mysql.user` table or by creating a user with an empty name with the [GRANT](#) statement:

```
GRANT ALL ON test.* TO ''@'localhost' ...
```

When specifying quoted values, quote database, table, column, and routine names as identifiers, using backticks (```). Quote hostnames, usernames, and passwords as strings, using single quotes (`'`).

Warning

If you allow anonymous users to connect to the MySQL server, you should also grant privileges to all local users as *user_name@localhost*. Otherwise, the anonymous user account for `localhost` in the `mysql.user` table (created during MySQL installation) is used when named users try to log in to the MySQL server from the local machine. For details, see [Section 5.7.5, “Access Control, Stage 1: Connection Verification”](#).

You can determine whether this applies to you by executing the following query, which lists any anonymous users:

```
SELECT Host, User FROM mysql.user WHERE User='';
```

If you want to delete the local anonymous user account to avoid the problem just described, use these statements:

```
DELETE FROM mysql.user WHERE Host='localhost' AND User='';
FLUSH PRIVILEGES;
```

[GRANT](#) supports hostnames up to 60 characters long. Database, table, column, and routine names can be up to 64 characters. Usernames can be up to 16 characters.

Note

The allowable length for usernames cannot be changed by altering the `mysql.user` table, and attempting to do so results in unpredictable behavior which may even make it impossible for users to log in to the MySQL server. You should never alter any of the tables in the `mysql` database in any manner whatsoever except by means of the procedure prescribed by MySQL AB that is described in [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

The privileges for a table, column, or routine are formed additively as the logical **OR** of the privileges at each of the privilege levels. For example, if the `mysql.user` table specifies that a user has a global [SELECT](#) privilege, the privilege cannot be denied by an entry at

the database, table, or column level.

The privileges for a column can be calculated as follows:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

In most cases, you grant rights to a user at only one of the privilege levels, so life is not normally this complicated. The details of the privilege-checking procedure are presented in [Section 5.7, “The MySQL Access Privilege System”](#).

If you grant privileges for a username/hostname combination that does not exist in the `mysql.user` table, an entry is added and remains there until deleted with a `DELETE` statement. In other words, `GRANT` may create `user` table entries, but `REVOKE` does not remove them; you must do that explicitly using `DROP USER` or `DELETE`.

Warning

If you create a new user but do not specify an `IDENTIFIED BY` clause, the user has no password. This is very insecure. However, you can enable the `NO_AUTO_CREATE_USER` SQL mode to prevent `GRANT` from creating a new user if it would otherwise do so, unless `IDENTIFIED BY` is given to provide the new user a non-empty password.

MySQL Enterprise

The MySQL Enterprise Monitor specifically guards against user accounts with no passwords. To find out more see <http://www.mysql.com/products/enterprise/advisors.html>.

If a new user is created or if you have global grant privileges, the user's password is set to the password specified by the `IDENTIFIED BY` clause, if one is given. If the user already had a password, this is replaced by the new one.

Passwords can also be set with the `SET PASSWORD` statement. See [Section 12.5.1.6, “SET PASSWORD Syntax”](#).

In the `IDENTIFIED BY` clause, the password should be given as the literal password value. It is unnecessary to use the `PASSWORD( )` function as it is for the `SET PASSWORD` statement. For example:

```
GRANT ... IDENTIFIED BY 'mypass';
```

If you do not want to send the password in clear text and you know the hashed value that `PASSWORD( )` would return for the password, you can specify the hashed value preceded by the keyword `PASSWORD`:

```
GRANT ...
IDENTIFIED BY PASSWORD '*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4';
```

In a C program, you can get the hashed value by using the `make_scrambled_password( )` C API function.

If you grant privileges for a database, an entry in the `mysql.db` table is created if needed. If all privileges for the database are removed with `REVOKE`, this entry is deleted.

The `SHOW DATABASES` privilege enables the account to see database names by issuing the `SHOW DATABASE` statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the `--skip-show-database` option.

MySQL Enterprise

The `SHOW DATABASES` privilege should be granted only to users who need to see all the databases on a MySQL server. Subscribers to the MySQL Enterprise Monitor are alerted when servers are started without the `--skip-show-database` option. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

If a user has no privileges for a table, the table name is not displayed when the user requests a list of tables (for example, with a `SHOW TABLES` statement).

The `WITH GRANT OPTION` clause gives the user the ability to give to other users any privileges the user has at the specified privilege level. You should be careful to whom you give the `GRANT OPTION` privilege, because two users with different privileges may be able to join privileges!

You cannot grant another user a privilege which you yourself do not have; the `GRANT OPTION` privilege enables you to assign only

those privileges which you yourself possess.

Be aware that when you grant a user the **GRANT OPTION** privilege at a particular privilege level, any privileges the user possesses (or may be given in the future) at that level can also be granted by that user to other users. Suppose that you grant a user the **INSERT** privilege on a database. If you then grant the **SELECT** privilege on the database and specify **WITH GRANT OPTION**, that user can give to other users not only the **SELECT** privilege, but also **INSERT**. If you then grant the **UPDATE** privilege to the user on the database, the user can grant **INSERT**, **SELECT**, and **UPDATE**.

For a non-administrative user, you should not grant the **ALTER** privilege globally or for the **mysql** database. If you do that, the user can try to subvert the privilege system by renaming tables!

The **MAX_QUERIES_PER_HOUR count**, **MAX_UPDATES_PER_HOUR count**, and **MAX_CONNECTIONS_PER_HOUR count** options limit the number of queries, updates, and logins a user can perform during any given one-hour period. (Queries for which results are served from the query cache do not count against the **MAX_QUERIES_PER_HOUR** limit.) If **count** is 0 (the default), this means that there is no limitation for that user.

The **MAX_USER_CONNECTIONS count** option limits the maximum number of simultaneous connections that the account can make. If **count** is 0 (the default), the **max_user_connections** system variable determines the number of simultaneous connections for the account.

Note: To specify any of these resource-limit options for an existing user without affecting existing privileges, use **GRANT USAGE ON *.* ... WITH MAX_...**

See [Section 5.8.4, “Limiting Account Resources”](#).

MySQL can check X509 certificate attributes in addition to the usual authentication that is based on the username and password. To specify SSL-related options for a MySQL account, use the **REQUIRE** clause of the **GRANT** statement. (For background information on the use of SSL with MySQL, see [Section 5.8.7, “Using Secure Connections”](#).)

There are a number of different possibilities for limiting connection types for a given account:

- **REQUIRE NONE** indicates that the account has no SSL or X509 requirements. This is the default if no SSL-related **REQUIRE** options are specified. Unencrypted connections are allowed if the username and password are valid. However, encrypted connections can also be used, at the client's option, if the client has the proper certificate and key files. That is, the client need not specify any SSL command options, in which case the connection will be unencrypted. To use an encrypted connection, the client must specify either the **--ssl-ca** option, or all three of the **--ssl-ca**, **--ssl-key**, and **--ssl-cert** options.
- The **REQUIRE SSL** option tells the server to allow only SSL-encrypted connections for the account.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret' REQUIRE SSL;
```

To connect, the client must specify the **--ssl-ca** option, and may additionally specify the **--ssl-key** and **--ssl-cert** options.

- **REQUIRE X509** means that the client must have a valid certificate but that the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret' REQUIRE X509;
```

To connect, the client must specify the **--ssl-ca**, **--ssl-key**, and **--ssl-cert** options. This is also true for **ISSUER** and **SUBJECT** because those **REQUIRE** options imply **X509**.

- **REQUIRE ISSUER 'issuer'** places the restriction on connection attempts that the client must present a valid X509 certificate issued by CA **'issuer'**. If the client presents a certificate that is valid but has a different issuer, the server rejects the connection. Use of X509 certificates always implies encryption, so the **SSL** option is unnecessary in this case.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/
    O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com';
```

Note that the **'issuer'** value should be entered as a single string.

- **REQUIRE SUBJECT 'subject'** places the restriction on connection attempts that the client must present a valid X509 certifi-

ate containing the subject *subject*. If the client presents a certificate that is valid but has a different subject, the server rejects the connection.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
    O=MySQL demo client certificate/
    CN=Tonu Samuel/Email=tonu@example.com';
```

Note that the '*subject*' value should be entered as a single string.

- **REQUIRE CIPHER** '*cipher*' is needed to ensure that ciphers and key lengths of sufficient strength are used. SSL itself can be weak if old algorithms using short encryption keys are used. Using this option, you can ask that a specific cipher method is used to allow a connection.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The **SUBJECT**, **ISSUER**, and **CIPHER** options can be combined in the **REQUIRE** clause like this:

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
    O=MySQL demo client certificate/
    CN=Tonu Samuel/Email=tonu@example.com'
  AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
    O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'
  AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The **AND** keyword is optional between **REQUIRE** options.

The order of the options does not matter, but no option can be specified twice.

When **mysql**d starts, all privileges are read into memory. For details, see [Section 5.7.7, “When Privilege Changes Take Effect”](#).

Note that if you are using table, column, or routine privileges for even one user, the server examines table, column, and routine privileges for all users and this slows down MySQL a bit. Similarly, if you limit the number of queries, updates, or connections for any users, the server must monitor these values.

The biggest differences between the standard SQL and MySQL versions of **GRANT** are:

- In MySQL, privileges are associated with the combination of a hostname and username and not with only a username.
- Standard SQL does not have global or database-level privileges, nor does it support all the privilege types that MySQL supports.
- MySQL does not support the standard SQL **UNDER** privilege.
- Standard SQL privileges are structured in a hierarchical manner. If you remove a user, all privileges the user has been granted are revoked. This is also true in MySQL if you use **DROP USER**. See [Section 12.5.1.2, “DROP USER Syntax”](#).
- In standard SQL, when you drop a table, all privileges for the table are revoked. In standard SQL, when you revoke a privilege, all privileges that were granted based on that privilege are also revoked. In MySQL, privileges can be dropped only with explicit **REVOKE** statements or by manipulating values stored in the MySQL grant tables.
- In MySQL, it is possible to have the **INSERT** privilege for only some of the columns in a table. In this case, you can still execute **INSERT** statements on the table, provided that you omit those columns for which you do not have the **INSERT** privilege. The omitted columns are set to their implicit default values if strict SQL mode is not enabled. In strict mode, the statement is rejected if any of the omitted columns have no default value. (Standard SQL requires you to have the **INSERT** privilege on all columns.) [Section 5.2.6, “SQL Modes”](#), discusses strict mode. [Section 10.1.4, “Data Type Default Values”](#), discusses implicit default values.

12.5.1.4. RENAME USER Syntax

```
RENAME USER old_user TO new_user
[, old_user TO new_user] ...
```

The **RENAME USER** statement renames existing MySQL accounts. To use it, you must have the global **CREATE USER** privilege or the **UPDATE** privilege for the **mysql** database. An error occurs if any old account does not exist or any new account exists. Each account is named using the same format as for the **GRANT** statement; for example, **'jeffrey'@'localhost'**. If you specify only the user-name part of the account name, a hostname part of **'%'** is used. For additional information about specifying account names, see [Section 12.5.1.3, “GRANT Syntax”](#).

RENAME USER does not automatically migrate any database objects that the user created, nor does it migrate any privileges that the user had prior to the renaming. This applies to tables, views, stored routines, triggers, and events.

12.5.1.5. REVOKE Syntax

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
ON [object_type] {tbl_name | * | *.* | db_name.*}
FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

The **REVOKE** statement enables system administrators to revoke privileges from MySQL accounts. To use **REVOKE**, you must have the **GRANT OPTION** privilege, and you must have the privileges that you are revoking.

Each account is named using the same format as for the **GRANT** statement; for example, **'jeffrey'@'localhost'**. If you specify only the username part of the account name, a hostname part of **'%'** is used. For additional information about specifying account names, see [Section 12.5.1.3, “GRANT Syntax”](#).

For details on the levels at which privileges exist, the allowable **priv_type** values, and the syntax for specifying users and passwords, see [Section 12.5.1.3, “GRANT Syntax”](#).

If the grant tables hold privilege rows that contain mixed-case database or table names and the **lower_case_table_names** system variable is set to a non-zero value, **REVOKE** cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (**GRANT** will not create such rows when **lower_case_table_names** is set, but such rows might have been created prior to setting the variable.)

To revoke all privileges, use the following syntax, which drops all global, database-, table-, and column-level privileges for the named user or users:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

To use this **REVOKE** syntax, you must have the global **CREATE USER** privilege or the **UPDATE** privilege for the **mysql** database.

REVOKE removes privileges, but does not drop **user** table entries. You must do that explicitly using **DELETE** or **DROP USER** (see [Section 12.5.1.2, “DROP USER Syntax”](#)).

12.5.1.6. SET PASSWORD Syntax

```
SET PASSWORD [FOR user] = PASSWORD('some password')
```

The **SET PASSWORD** statement assigns a password to an existing MySQL user account.

With no **FOR** clause, this statement sets the password for the current user. Any client that has connected to the server using a non-anonymous account can change the password for that account.

With a **FOR** clause, this statement sets the password for a specific account on the current server host. Only clients that have the **UPDATE** privilege for the **mysql** database can do this. The **user** value should be given in **user_name@host_name** format, where **user_name** and **host_name** are exactly as they are listed in the **User** and **Host** columns of the **mysql.user** table entry. For example, if you had an entry with **User** and **Host** column values of **'bob'** and **'%.loc.gov'**, you would write the statement like this:

```
SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

That is equivalent to the following statements:

```
UPDATE mysql.user SET Password=PASSWORD('newpass')
WHERE User='bob' AND Host='%.loc.gov';
FLUSH PRIVILEGES;
```

Note

If you are connecting to a MySQL 4.1 or later server using a pre-4.1 client program, do not use the preceding `SET PASSWORD` or `UPDATE` statement without reading [Section 5.7.9, “Password Hashing as of MySQL 4.1”](#), first. The password format changed in MySQL 4.1, and under certain circumstances it is possible that if you change your password, you might not be able to connect to the server afterward.

You can see which account the server authenticated you as by executing `SELECT CURRENT_USER()`.

MySQL Enterprise

For automated notification of users without passwords, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

12.5.2. Table Maintenance Statements

12.5.2.1. ANALYZE TABLE Syntax

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

`ANALYZE TABLE` analyzes and stores the key distribution for a table. During the analysis, the table is locked with a read lock for `MyISAM`. For `InnoDB` the table is locked with a write lock. This statement works with `MyISAM`, and `InnoDB` tables. For `MyISAM` tables, this statement is equivalent to using `myisamchk --analyze`.

For more information on how the analysis works within `InnoDB`, see [Section 13.5.16, “Restrictions on InnoDB Tables”](#).

MySQL Enterprise

For expert advice on optimizing tables subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

MySQL uses the stored key distribution to decide the order in which tables should be joined when you perform a join on something other than a constant.

This statement requires `SELECT` and `INSERT` privileges for the table.

`ANALYZE TABLE` returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always <code>analyze</code>
Msg_type	One of <code>status</code> , <code>error</code> , <code>info</code> , or <code>warning</code>
Msg_text	The message

You can check the stored key distribution with the `SHOW INDEX` statement. See [Section 12.5.4.18, “SHOW INDEX Syntax”](#).

If the table has not changed since the last `ANALYZE TABLE` statement, the table is not analyzed again.

By default, `ANALYZE TABLE` statements are written to the binary log so that such statements used on a MySQL server acting as a replication master will be replicated to replication slaves. Logging can be suppressed with the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

12.5.2.2. BACKUP TABLE Syntax

```
BACKUP TABLE tbl_name [, tbl_name] ... TO '/path/to/backup/directory'
```

Note

This statement is deprecated. We are working on a better replacement for it that will provide online backup capabilities. In the meantime, the `mysqlhotcopy` script can be used instead.

`BACKUP TABLE` copies to the backup directory the minimum number of table files needed to restore the table, after flushing any buf-

ferred changes to disk. The statement works only for **MyISAM** tables. It copies the `.frm` definition and `.MYD` data files. The `.MYI` index file can be rebuilt from those two files. The directory should be specified as a full pathname. To restore the table, use **RESTORE TABLE**.

During the backup, a read lock is held for each table, one at time, as they are being backed up. If you want to back up several tables as a snapshot (preventing any of them from being changed during the backup operation), issue a **LOCK TABLES** statement first, to obtain a read lock for all tables in the group.

BACKUP TABLE returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always backup
Msg_type	One of status , error , info , or warning
Msg_text	The message

12.5.2.3. CHECK TABLE Syntax

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...
option = {FOR UPGRADE | QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

CHECK TABLE checks a table or tables for errors. **CHECK TABLE** works for **MyISAM**, **InnoDB**, and **ARCHIVE** tables. Starting with MySQL 5.1.9, **CHECK** is also valid for **CSV** tables, see [Section 13.11, “The CSV Storage Engine”](#). For **MyISAM** tables, the key statistics are updated as well.

CHECK TABLE can also check views for problems, such as tables that are referenced in the view definition that no longer exist.

CHECK TABLE returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always check
Msg_type	One of status , error , info , or warning
Msg_text	The message

Note that the statement might produce many rows of information for each checked table. The last row has a **Msg_type** value of **status** and the **Msg_text** normally should be **OK**. If you don't get **OK**, or **Table is already up to date** you should normally run a repair of the table. See [Section 5.9.4, “Table Maintenance and Crash Recovery”](#). **Table is already up to date** means that the storage engine for the table indicated that there was no need to check the table.

The **FOR UPGRADE** option checks whether the named tables are compatible with the current version of MySQL. This option was added in MySQL 5.1.7. With **FOR UPGRADE**, the server checks each table to determine whether there have been any incompatible changes in any of the table's data types or indexes since the table was created. If not, the check succeeds. Otherwise, if there is a possible incompatibility, the server runs a full check on the table (which might take some time). If the full check succeeds, the server marks the table's `.frm` file with the current MySQL version number. Marking the `.frm` file ensures that further checks for the table with the same version of the server will be fast.

Incompatibilities might occur because the storage format for a data type has changed or because its sort order has changed. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.

Currently, **FOR UPGRADE** discovers these incompatibilities:

- The indexing order for end-space in **TEXT** columns for **InnoDB** and **MyISAM** tables changed between MySQL 4.1 and 5.0.
- The storage method of the new **DECIMAL** data type changed between MySQL 5.0.3 and 5.0.5.

The other check options that can be given are shown in the following table. These options apply only to checking **MyISAM** tables and are ignored for **InnoDB** tables and views.

Type	Meaning
QUICK	Do not scan the rows to check for incorrect links.
FAST	Check only tables that have not been closed properly.
CHANGED	Check only tables that have been changed since the last check or that have not been closed properly.
MEDIUM	Scan rows to verify that deleted links are valid. This also calculates a key checksum for the rows and verifies this with a calculated checksum for the keys.
EXTENDED	Do a full key lookup for all keys for each row. This ensures that the table is 100% consistent, but takes a long time.

If none of the options **QUICK**, **MEDIUM**, or **EXTENDED** are specified, the default check type for dynamic-format **MyISAM** tables is **MEDIUM**. This has the same result as running `myisamchk --medium-check tbl_name` on the table. The default check type also is **MEDIUM** for static-format **MyISAM** tables, unless **CHANGED** or **FAST** is specified. In that case, the default is **QUICK**. The row scan is skipped for **CHANGED** and **FAST** because the rows are very seldom corrupted.

You can combine check options, as in the following example that does a quick check on the table to determine whether it was closed properly:

```
CHECK TABLE test_table FAST QUICK;
```

Note

In some cases, **CHECK TABLE** changes the table. This happens if the table is marked as “corrupted” or “not closed properly” but **CHECK TABLE** does not find any problems in the table. In this case, **CHECK TABLE** marks the table as okay.

If a table is corrupted, it is most likely that the problem is in the indexes and not in the data part. All of the preceding check types check the indexes thoroughly and should thus find most errors.

If you just want to check a table that you assume is okay, you should use no check options or the **QUICK** option. The latter should be used when you are in a hurry and can take the very small risk that **QUICK** does not find an error in the data file. (In most cases, under normal usage, MySQL should find any error in the data file. If this happens, the table is marked as “corrupted” and cannot be used until it is repaired.)

FAST and **CHANGED** are mostly intended to be used from a script (for example, to be executed from **cron**) if you want to check tables from time to time. In most cases, **FAST** is to be preferred over **CHANGED**. (The only case when it is not preferred is when you suspect that you have found a bug in the **MyISAM** code.)

EXTENDED is to be used only after you have run a normal check but still get strange errors from a table when MySQL tries to update a row or find a row by key. This is very unlikely if a normal check has succeeded.

Use of **CHECK TABLE ... EXTENDED** might influence the execution plan generated by the query optimizer.

Some problems reported by **CHECK TABLE** cannot be corrected automatically:

- **Found row where the auto_increment column has the value 0.**

This means that you have a row in the table where the **AUTO_INCREMENT** index column contains the value 0. (It is possible to create a row where the **AUTO_INCREMENT** column is 0 by explicitly setting the column to 0 with an **UPDATE** statement.)

This is not an error in itself, but could cause trouble if you decide to dump the table and restore it or do an **ALTER TABLE** on the table. In this case, the **AUTO_INCREMENT** column changes value according to the rules of **AUTO_INCREMENT** columns, which could cause problems such as a duplicate-key error.

To get rid of the warning, simply execute an **UPDATE** statement to set the column to some value other than 0.

- If **CHECK TABLE** finds a problem for an **InnoDB** table, the server shuts down to prevent error propagation. Details of the error will be written to the error log.

12.5.2.4. CHECKSUM TABLE Syntax

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

CHECKSUM TABLE reports a table checksum.

With **QUICK**, the live table checksum is reported if it is available, or **NULL** otherwise. This is very fast. A live checksum is enabled by specifying the **CHECKSUM=1** table option when you create the table; currently, this is supported only for **MyISAM** tables. See [Section 12.1.10, “CREATE TABLE Syntax”](#).

With **EXTENDED**, the entire table is read row by row and the checksum is calculated. This can be very slow for large tables.

If neither **QUICK** nor **EXTENDED** is specified, MySQL returns a live checksum if the table storage engine supports it and scans the table otherwise.

For a non-existent table, **CHECKSUM TABLE** returns **NULL** and generates a warning.

The checksum value depends on the table row format. If the row format changes, the checksum also changes. For example, the storage format for **VARCHAR** changed between MySQL 4.1 and 5.0, so if a 4.1 table is upgraded to MySQL 5.0, the checksum value may change.

12.5.2.5. OPTIMIZE TABLE Syntax

```
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

OPTIMIZE TABLE should be used if you have deleted a large part of a table or if you have made many changes to a table with variable-length rows (tables that have **VARCHAR**, **VARBINARY**, **BLOB**, or **TEXT** columns). Deleted rows are maintained in a linked list and subsequent **INSERT** operations reuse old row positions. You can use **OPTIMIZE TABLE** to reclaim the unused space and to defragment the data file.

This statement requires **SELECT** and **INSERT** privileges for the table.

In most setups, you need not run **OPTIMIZE TABLE** at all. Even if you do a lot of updates to variable-length rows, it is not likely that you need to do this more than once a week or month and only on certain tables.

OPTIMIZE TABLE works *only* for **MyISAM**, **InnoDB**, and **ARCHIVE** tables. It does *not* work for tables created using any other storage engine, including **NDB** Disk Data tables.

For **MyISAM** tables, **OPTIMIZE TABLE** works as follows:

1. If the table has deleted or split rows, repair the table.
2. If the index pages are not sorted, sort them.
3. If the table's statistics are not up to date (and the repair could not be accomplished by sorting the index), update them.

For **InnoDB** tables, **OPTIMIZE TABLE** is mapped to **ALTER TABLE**, which rebuilds the table to update index statistics and free unused space in the clustered index.

You can make **OPTIMIZE TABLE** work on other storage engines by starting **mysqld** with the **--skip-new** or **--safe-mode** option. In this case, **OPTIMIZE TABLE** is just mapped to **ALTER TABLE**.

OPTIMIZE TABLE returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always optimize
Msg_type	One of status , error , info , or warning
Msg_text	The message

Note that MySQL locks the table during the time **OPTIMIZE TABLE** is running.

By default, **OPTIMIZE TABLE** statements are written to the binary log so that such statements used on a MySQL server acting as a replication master will be replicated to replication slaves. Logging can be suppressed with the optional **NO_WRITE_TO_BINLOG** keyword or its alias **LOCAL**.

OPTIMIZE TABLE does not sort R-tree indexes, such as spatial indexes on **POINT** columns. (Bug#23578)

12.5.2.6. REPAIR TABLE Syntax

```
REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE
      tbl_name [, tbl_name] ... [QUICK] [EXTENDED] [USE_FRM]
```

REPAIR TABLE repairs a possibly corrupted table. By default, it has the same effect as `myisamchk --recover tbl_name`. **REPAIR TABLE** works for **MyISAM** and for **ARCHIVE** tables. Starting with MySQL 5.1.9, **REPAIR** is also valid for **CSV** tables. See Section 13.4, “The MyISAM Storage Engine”, and Section 13.10, “The ARCHIVE Storage Engine”, and Section 13.11, “The CSV Storage Engine”

This statement requires **SELECT** and **INSERT** privileges for the table.

Normally, you should never have to run this statement. However, if disaster strikes, **REPAIR TABLE** is very likely to get back all your data from a **MyISAM** table. If your tables become corrupted often, you should try to find the reason for it, to eliminate the need to use **REPAIR TABLE**. See Section B.1.4.2, “What to Do If MySQL Keeps Crashing”, and Section 13.4.4, “MyISAM Table Problems”.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to filesystem errors.

Warning

If the server dies during a **REPAIR TABLE** operation, it is essential after restarting it that you immediately execute another **REPAIR TABLE** statement for the table before performing any other operations on it. (It is always a good idea to start by making a backup.) In the worst case, you might have a new clean index file without information about the data file, and then the next operation you perform could overwrite the data file. This is an unlikely but possible scenario.

REPAIR TABLE returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always repair
Msg_type	One of status , error , info , or warning
Msg_text	The message

The **REPAIR TABLE** statement might produce many rows of information for each repaired table. The last row has a **Msg_type** value of **status** and **Msg_text** normally should be **OK**. If you do not get **OK**, you should try repairing the table with `myisamchk -safe-recover`. (**REPAIR TABLE** does not yet implement all the options of `myisamchk`.) With `myisamchk -safe-recover`, you can also use options that **REPAIR TABLE** does not support, such as `--max-record-length`.

If **QUICK** is given, **REPAIR TABLE** tries to repair only the index tree. This type of repair is like that done by `myisamchk -recover --quick`.

If you use **EXTENDED**, MySQL creates the index row by row instead of creating one index at a time with sorting. This type of repair is like that done by `myisamchk -safe-recover`.

There is also a **USE_FRM** mode available for **REPAIR TABLE**. Use this if the **.MYI** index file is missing or if its header is corrupted. In this mode, MySQL re-creates the **.MYI** file using information from the **.frm** file. This kind of repair cannot be done with `myisamchk`.

Note

Use this mode *only* if you cannot use regular **REPAIR** modes. The **.MYI** header contains important table metadata (in par-

ticular, current [AUTO_INCREMENT](#) value and [Delete link](#)) that are lost in `REPAIR ... USE_FRM`. Don't use `USE_FRM` if the table is compressed because this information is also stored in the `.MYI` file.

Caution

Do not use `USE_FRM` if your table was created by a different version of the MySQL server than the one you are currently running. Doing so risks the loss of all rows in the table. It is particularly dangerous to use `USE_FRM` after the server returns this message:

```
Table upgrade required. Please do
"REPAIR TABLE `tbl_name`" to fix it!
```

By default, `REPAIR TABLE` statements are written to the binary log so that such statements used on a MySQL server acting as a replication master will be replicated to replication slaves. Logging can be suppressed with the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

12.5.2.7. RESTORE TABLE Syntax

```
RESTORE TABLE tbl_name [, tbl_name] ... FROM '/path/to/backup/directory'
```

`RESTORE TABLE` restores the table or tables from a backup that was made with `BACKUP TABLE`. The directory should be specified as a full pathname.

Existing tables are not overwritten; if you try to restore over an existing table, an error occurs. Just as for `BACKUP TABLE`, `RESTORE TABLE` currently works only for `MyISAM` tables. Restored tables are not replicated from master to slave.

The backup for each table consists of its `.frm` format file and `.MYD` data file. The restore operation restores those files, and then uses them to rebuild the `.MYI` index file. Restoring takes longer than backing up due to the need to rebuild the indexes. The more indexes the table has, the longer it takes.

`RESTORE TABLE` returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always <code>restore</code>
Msg_type	One of <code>status</code> , <code>error</code> , <code>info</code> , or <code>warning</code>
Msg_text	The message

12.5.3. SET Syntax

```
SET variable_assignment [, variable_assignment] ...

variable_assignment:
    user_var_name = expr
    | [GLOBAL | SESSION] system_var_name = expr
    | [@@global. | @@session. | @@] system_var_name = expr
```

The `SET` statement assigns values to different types of variables that affect the operation of the server or your client. Older versions of MySQL employed `SET OPTION`, but this syntax is deprecated in favor of `SET` without `OPTION`.

This section describes use of `SET` for assigning values to system variables or user variables. For general information about these types of variables, see [Section 5.2.3, “System Variables”](#), and [Section 8.4, “User-Defined Variables”](#). System variables also can be set at server startup, as described in [Section 5.2.4, “Using System Variables”](#).

Some variants of `SET` syntax are used in other contexts:

- `SET PASSWORD` assigns account passwords. See [Section 12.5.1.6, “SET PASSWORD Syntax”](#).
- `SET TRANSACTION ISOLATION LEVEL` sets the isolation level for transaction processing. See [Section 12.4.6, “SET TRANSACTION Syntax”](#).
- `SET` is used within stored routines to assign values to local routine variables. See [Section 19.2.7.2, “Variable SET Statement”](#).

The following discussion shows the different **SET** syntaxes that you can use to set variables. The examples use the **=** assignment operator, but the **:=** operator also is allowable.

A user variable is written as **@var_name** and can be set as follows:

```
SET @var_name = expr;
```

Many system variables are dynamic and can be changed while the server runs by using the **SET** statement. For a list, see [Section 5.2.4.2, “Dynamic System Variables”](#). To change a system variable with **SET**, refer to it as **var_name**, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by **GLOBAL** or **@@global..** The **SUPER** privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by **SESSION**, **@@session.**, or **@@**. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- **LOCAL** and **@@local.** are synonyms for **SESSION** and **@@session..**
- If no modifier is present, **SET** changes the session variable.

MySQL Enterprise

The MySQL Enterprise Monitor makes extensive use of system variables to determine the state of your server. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

A **SET** statement can contain multiple variable assignments, separated by commas. If you set several system variables, the most recent **GLOBAL** or **SESSION** modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

When you assign a value to a system variable with **SET**, you cannot use suffix letters in the value (as can be done with startup options). However, the value can take the form of an expression:

```
SET sort_buffer_size = 10 * 1024 * 1024;
```

The **@@var_name** syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the **SET GLOBAL** statement).

To prevent incorrect usage, MySQL produces an error if you use **SET GLOBAL** with a variable that can only be used with **SET SESSION** or if you do not specify **GLOBAL** (or **@@global.**) when setting a global variable.

To set a **SESSION** variable to the **GLOBAL** value or a **GLOBAL** value to the compiled-in MySQL default value, use the **DEFAULT** keyword. For example, the following two statements are identical in setting the session value of **max_join_size** to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to **DEFAULT**. In such cases, use of **DEFAULT** results in an error.

You can refer to the values of specific global or session system variables in expressions by using one of the @@-modifiers. For example, you can retrieve values in a `SELECT` statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as `@@var_name` (that is, when you do not specify `@@global.` or `@@session.`), MySQL returns the session value if it exists and the global value otherwise. (This differs from `SET @@var_name = value`, which always refers to the session value.)

To display system variables names and values, use the `SHOW VARIABLES` statement. (See [Section 12.5.4.31, “SHOW VARIABLES Syntax”](#).)

The following list describes options that have non-standard syntax or that are not described in the list of system variables found in [Section 5.2.3, “System Variables”](#). Although the options described here are not displayed by `SHOW VARIABLES`, you can obtain their values with `SELECT` (with the exception of `CHARACTER SET` and `SET NAMES`). For example:

```
mysql> SELECT @@AUTOCOMMIT;
+-----+
| @@AUTOCOMMIT |
+-----+
|             1 |
+-----+
```

The lettercase of these options does not matter.

- `AUTOCOMMIT = {0 | 1}`

Set the autocommit mode. If set to 1, all changes to a table take effect immediately. If set to 0 you have to use `COMMIT` to accept a transaction or `ROLLBACK` to cancel it. By default, client connections begin with `AUTOCOMMIT` set to 1. If you change `AUTOCOMMIT` mode from 0 to 1, MySQL performs an automatic `COMMIT` of any open transaction. Another way to begin a transaction is to use a `START TRANSACTION` or `BEGIN` statement. See [Section 12.4.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#).

- `BIG_TABLES = {0 | 1}`

If set to 1, all temporary tables are stored on disk rather than in memory. This is a little slower, but the error `The table tbl_name is full` does not occur for `SELECT` operations that require a large temporary table. The default value for a new connection is 0 (use in-memory temporary tables). Normally, you should never need to set this variable, because in-memory tables are automatically converted to disk-based tables as required.

Note

This variable was formerly named `SQL_BIG_TABLES`.

- `CHARACTER SET {charset_name | DEFAULT}`

This maps all strings from and to the client with the given mapping. You can add new mappings by editing `sql/convert.cc` in the MySQL source distribution. `SET CHARACTER SET` sets three session system variables: `character_set_client` and `character_set_results` are set to the given character set, and `character_set_connection` to the value of `character_set_database`. See [Section 9.4, “Connection Character Sets and Collations”](#).

The default mapping can be restored by using the value `DEFAULT`. The default depends on the server configuration.

Note that the syntax for `SET CHARACTER SET` differs from that for setting most other options.

- `FOREIGN_KEY_CHECKS = {0 | 1}`

If set to 1 (the default), foreign key constraints for `InnoDB` tables are checked. If set to 0, they are ignored. Disabling foreign key checking can be useful for reloading `InnoDB` tables in an order different from that required by their parent/child relationships. See [Section 13.5.6.4, “FOREIGN KEY Constraints”](#).

Setting `FOREIGN_KEY_CHECKS` to 0 also affects data definition statements: `DROP SCHEMA` drops a schema even if it contains tables that have foreign keys that are referred to by tables outside the schema, and `DROP TABLE` drops tables that have foreign keys that are referred to by other tables.

Note

Setting `FOREIGN_KEY_CHECKS` to 1 does not trigger a scan of the existing table data. Therefore, rows added to the table while `FOREIGN_KEY_CHECKS=0` will not be verified for consistency.

- `IDENTITY = value`

This variable is a synonym for the `LAST_INSERT_ID` variable. It exists for compatibility with other database systems. You can read its value with `SELECT @@IDENTITY`, and set it using `SET IDENTITY`.

- `INSERT_ID = value`

Set the value to be used by the following `INSERT` or `ALTER TABLE` statement when inserting an `AUTO_INCREMENT` value. This is mainly used with the binary log.

- `LAST_INSERT_ID = value`

Set the value to be returned from `LAST_INSERT_ID()`. This is stored in the binary log when you use `LAST_INSERT_ID()` in a statement that updates a table. Setting this variable does not update the value returned by the `mysql_insert_id()` C API function.

- `NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT}`

`SET NAMES` sets the three session system variables `character_set_client`, `character_set_connection`, and `character_set_results` to the given character set. Setting `character_set_connection` to `charset_name` also sets `collation_connection` to the default collation for `charset_name`. The optional `COLLATE` clause may be used to specify a collation explicitly. See [Section 9.4, “Connection Character Sets and Collations”](#).

The default mapping can be restored by using a value of `DEFAULT`. The default depends on the server configuration.

Note that the syntax for `SET NAMES` differs from that for setting most other options.

- `ONE_SHOT`

This option is a modifier, not a variable. It can be used to influence the effect of variables that set the character set, the collation, and the time zone. `ONE_SHOT` is primarily used for replication purposes: `mysqlbinlog` uses `SET ONE_SHOT` to modify temporarily the values of character set, collation, and time zone variables to reflect at rollforward what they were originally. `ONE_SHOT` is for internal use only and is deprecated for MySQL 5.0 and up.

You cannot use `ONE_SHOT` with other than the allowed set of variables; if you try, you get an error like this:

```
mysql> SET ONE_SHOT max_allowed_packet = 1;
ERROR 1382 (HY000): The 'SET ONE_SHOT' syntax is reserved for purposes
internal to the MySQL server
```

If `ONE_SHOT` is used with the allowed variables, it changes the variables as requested, but only for the next non-`SET` statement. After that, the server resets all character set, collation, and time zone-related system variables to their previous values. Example:

```
mysql> SET ONE_SHOT character_set_connection = latin5;
mysql> SET ONE_SHOT collation_connection = latin5_turkish_ci;
mysql> SHOW VARIABLES LIKE '%_connection';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_connection | latin5 |
| collation_connection | latin5_turkish_ci |
+-----+-----+

mysql> SHOW VARIABLES LIKE '%_connection';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_connection | latin1 |
| collation_connection | latin1_swedish_ci |
+-----+-----+
```

- `SQL_AUTO_IS_NULL = {0 | 1}`

If set to 1 (the default), you can find the last inserted row for a table that contains an `AUTO_INCREMENT` column by using the following construct:

```
WHERE auto_increment_column IS NULL
```

This behavior is used by some ODBC programs, such as Access.

- `SQL_BIG_SELECTS = {0 | 1}`

If set to 0, MySQL aborts `SELECT` statements that are likely to take a very long time to execute (that is, statements for which the optimizer estimates that the number of examined rows exceeds the value of `max_join_size`). This is useful when an inadvisable `WHERE` statement has been issued. The default value for a new connection is 1, which allows all `SELECT` statements.

If you set the `max_join_size` system variable to a value other than `DEFAULT`, `SQL_BIG_SELECTS` is set to 0.

- `SQL_BUFFER_RESULT = {0 | 1}`

If set to 1, `SQL_BUFFER_RESULT` forces results from `SELECT` statements to be put into temporary tables. This helps MySQL free the table locks early and can be beneficial in cases where it takes a long time to send results to the client. The default value is 0.

- `SQL_LOG_BIN = {0 | 1}`

If set to 0, no logging is done to the binary log for the client. The client must have the `SUPER` privilege to set this option. The default value is 1.

- `SQL_LOG_OFF = {0 | 1}`

If set to 1, no logging is done to the general query log for this client. The client must have the `SUPER` privilege to set this option. The default value is 0.

- `SQL_LOG_UPDATE = {0 | 1}`

This variable is deprecated, and is mapped to `SQL_LOG_BIN`.

- `SQL_NOTES = {0 | 1}`

If set to 1 (the default), warnings of `Note` level are recorded. If set to 0, `Note` warnings are suppressed. `mysqldump` includes output to set this variable to 0 so that reloading the dump file does not produce warnings for events that do not affect the integrity of the reload operation.

- `SQL_QUOTE_SHOW_CREATE = {0 | 1}`

If set to 1 (the default), the server quotes identifiers for `SHOW CREATE TABLE` and `SHOW CREATE DATABASE` statements. If set to 0, quoting is disabled. This option is enabled by default so that replication works for identifiers that require quoting. See [Section 12.5.4.9, “SHOW CREATE TABLE Syntax”](#), and [Section 12.5.4.6, “SHOW CREATE DATABASE Syntax”](#).

- `SQL_SAFE_UPDATES = {0 | 1}`

If set to 1, MySQL aborts `UPDATE` or `DELETE` statements that do not use a key in the `WHERE` clause or a `LIMIT` clause. This makes it possible to catch `UPDATE` or `DELETE` statements where keys are not used properly and that would probably change or delete a large number of rows. The default value is 0.

- `SQL_SELECT_LIMIT = {value | DEFAULT}`

The maximum number of rows to return from `SELECT` statements. The default value for a new connection is “unlimited.” If you have changed the limit, the default value can be restored by using a `SQL_SELECT_LIMIT` value of `DEFAULT`.

If a `SELECT` has a `LIMIT` clause, the `LIMIT` takes precedence over the value of `SQL_SELECT_LIMIT`.

`SQL_SELECT_LIMIT` does not apply to `SELECT` statements executed within stored routines. It also does not apply to `SELECT` statements that do not produce a result set to be returned to the client. These include `SELECT` statements in subqueries, `CREATE TABLE ... SELECT`, and `INSERT INTO ... SELECT`.

- `SQL_WARNINGS = {0 | 1}`

This variable controls whether single-row **INSERT** statements produce an information string if warnings occur. The default is 0. Set the value to 1 to produce an information string.

- **TIMESTAMP** = {*timestamp_value* | DEFAULT}

Set the time for this client. This is used to get the original timestamp if you use the binary log to restore rows. *timestamp_value* should be a Unix epoch timestamp, not a MySQL timestamp.

SET TIMESTAMP affects the value returned by **NOW()** but not by **SYSDATE()**. This means that timestamp settings in the binary log have no effect on invocations of **SYSDATE()**. The server can be started with the **--sysdate-is-now** option to cause **SYS-DATE()** to be an alias for **NOW()**, in which case **SET TIMESTAMP** affects both functions.

- **UNIQUE_CHECKS** = {0 | 1}

If set to 1 (the default), uniqueness checks for secondary indexes in **InnoDB** tables are performed. If set to 0, storage engines are allowed to assume that duplicate keys are not present in input data. If you know for certain that your data does not contain uniqueness violations, you can set this to 0 to speed up large table imports to **InnoDB**.

Note that setting this variable to 0 does not *require* storage engines to ignore duplicate keys. An engine is still allowed to check for them and issue duplicate-key errors if it detects them.

12.5.4. SHOW Syntax

SHOW has many forms that provide information about databases, tables, columns, or status information about the server. This section describes those following:

```
SHOW AUTHORS
SHOW CHARACTER SET [like_or_where]
[like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
SHOW CONTRIBUTORS
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION funcname
SHOW CREATE PROCEDURE procname
SHOW CREATE TABLE tbl_name
SHOW CREATE TRIGGER trigger_name
SHOW CREATE VIEW view_name
SHOW DATABASES [like_or_where]
SHOW ENGINE engine_name {STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW [FULL] EVENTS
SHOW FUNCTION CODE sp_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW INNODB STATUS
SHOW OPEN TABLES [FROM db_name] [like_or_where]
SHOW PROCEDURE CODE sp_name
SHOW PROCEDURE STATUS [like_or_where]
SHOW PLUGINS
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW SCHEDULER STATUS
SHOW [GLOBAL | SESSION] STATUS [like_or_where]
SHOW TABLE STATUS [FROM db_name] [like_or_where]
SHOW TABLES [FROM db_name] [like_or_where]
SHOW TRIGGERS [FROM db_name] [like_or_where]
SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
SHOW WARNINGS [LIMIT [offset,] row_count]

like_or_where:
    LIKE 'pattern'
    | WHERE expr
```

The **SHOW** statement also has forms that provide information about replication master and slave servers and are described in [Section 12.6, “Replication Statements”](#):

```
SHOW BINARY LOGS
SHOW BINLOG EVENTS
SHOW MASTER STATUS
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
```

If the syntax for a given `SHOW` statement includes a `LIKE 'pattern'` part, `'pattern'` is a string that can contain the SQL `'%'` and `'_'` wildcard characters. The pattern is useful for restricting statement output to matching values.

Several `SHOW` statements also accept a `WHERE` clause that provides more flexibility in specifying which rows to display. See [Section 23.27, “Extensions to SHOW Statements”](#).

Many MySQL APIs (such as PHP) allow you to treat the result returned from a `SHOW` statement as you would a result set from a `SELECT`; see [Chapter 25, APIs and Libraries](#), or your API documentation for more information. In addition, you can work in SQL with results from queries on tables in the `INFORMATION_SCHEMA` database, which you cannot easily do with results from `SHOW` statements. See [Chapter 23, The INFORMATION_SCHEMA Database](#).

12.5.4.1. SHOW AUTHORS Syntax

```
SHOW AUTHORS
```

The `SHOW AUTHORS` statement displays information about the people who work on MySQL. For each author, it displays `Name`, `Location`, and `Comment` values.

This statement was added in MySQL 5.1.3.

12.5.4.2. SHOW CHARACTER SET Syntax

```
SHOW CHARACTER SET
[LIKE 'pattern' | WHERE expr]
```

The `SHOW CHARACTER SET` statement shows all available character sets. The `LIKE` clause, if present, indicates which character set names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#). For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
```

Charset	Description	Default collation	Maxlen
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	1
latin7	ISO 8859-13 Baltic	latin7_general_ci	1

The `Maxlen` column shows the maximum number of bytes required to store one character.

12.5.4.3. SHOW COLLATION Syntax

```
SHOW COLLATION
[LIKE 'pattern' | WHERE expr]
```

The output from `SHOW COLLATION` includes all available character sets. The `LIKE` clause, if present, indicates which collation names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#). For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
```

Collation	Charset	Id	Default	Compiled	Sortlen
latin1_german1_ci	latin1	5			0
latin1_swedish_ci	latin1	8	Yes	Yes	0
latin1_danish_ci	latin1	15			0
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	0
latin1_general_ci	latin1	48			0
latin1_general_cs	latin1	49			0
latin1_spanish_ci	latin1	94			0

The `Default` column indicates whether a collation is the default for its character set. `Compiled` indicates whether the character set is compiled into the server. `Sortlen` is related to the amount of memory required to sort strings expressed in the character set.

12.5.4.4. SHOW COLUMNS Syntax

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name]
[LIKE 'pattern' | WHERE expr]
```

SHOW COLUMNS displays information about the columns in a given table. It also works for views. The **LIKE** clause, if present, indicates which column names to match. The **WHERE** clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#).

The **FULL** keyword causes the output to include the privileges you have as well as any per-column comments for each column.

You can use *db_name.tbl_name* as an alternative to the *tbl_name FROM db_name* syntax. In other words, these two statements are equivalent:

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;
mysql> SHOW COLUMNS FROM mydb.mytable;
```

SHOW FIELDS is a synonym for **SHOW COLUMNS**. You can also list a table's columns with the `mysqlshow db_name tbl_name` command.

The **DESCRIBE** statement provides information similar to **SHOW COLUMNS**. See [Section 12.3.1, “DESCRIBE Syntax”](#).

12.5.4.5. SHOW CONTRIBUTORS Syntax

```
SHOW CONTRIBUTORS
```

The **SHOW CONTRIBUTORS** statement displays information about the people who contribute to MySQL source or to causes that MySQL AB supports. For each contributor, it displays **Name**, **Location**, and **Comment** values.

This statement was added in MySQL 5.1.12.

12.5.4.6. SHOW CREATE DATABASE Syntax

```
SHOW CREATE {DATABASE | SCHEMA} db_name
```

Shows the **CREATE DATABASE** statement that creates the given database. **SHOW CREATE SCHEMA** is a synonym for **SHOW CREATE DATABASE**.

```
mysql> SHOW CREATE DATABASE test\G
***** 1. row *****
      Database: test
Create Database: CREATE DATABASE `test`
                  /*!40100 DEFAULT CHARACTER SET latin1 */

mysql> SHOW CREATE SCHEMA test\G
***** 1. row *****
      Database: test
Create Database: CREATE DATABASE `test`
                  /*!40100 DEFAULT CHARACTER SET latin1 */
```

SHOW CREATE DATABASE quotes table and column names according to the value of the `SQL_QUOTE_SHOW_CREATE` option. See [Section 12.5.3, “SET Syntax”](#).

12.5.4.7. SHOW CREATE EVENT

```
SHOW CREATE EVENT event_name
```

This statement displays the **CREATE EVENT** statement needed to re-create a given event. For example (using the same event `e_daily` defined and then altered in [Section 12.5.4.16, “SHOW EVENTS”](#)):

```
mysql> SHOW CREATE EVENT test.e_daily\G
***** 1. row *****
      Event: e_daily
Create Event: CREATE EVENT e_daily
              ON SCHEDULE EVERY 1 DAY
              STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
```

```

        ENABLE
        COMMENT 'Saves total number of sessions and
                clears the table once per day.'
    DO
        BEGIN
            INSERT INTO site_activity.totals (when, total)
            SELECT CURRENT_TIMESTAMP, COUNT(*)
            FROM site_activity.sessions;
            DELETE FROM site_activity.sessions;
        END
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci

```

`character_set_client` is the session value of the `character_set_client` system variable when the event was created. `collation_connection` is the session value of the `collation_connection` system variable when the event was created. `Database Collation` is the collation of the database with which the event is associated. These columns were added in MySQL 5.1.21.

Note that the output reflects the current status of the event (`ENABLE`) rather than the status with which it was created.

This statement was implemented in MySQL 5.1.6.

12.5.4.8. SHOW CREATE PROCEDURE and SHOW CREATE FUNCTION Syntax

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
```

These statements are MySQL extensions. Similar to `SHOW CREATE TABLE`, they return the exact string that can be used to re-create the named routine. The statements require that you be the owner of the routine or have `SELECT` access to the `mysql.proc` table.

```

mysql> SHOW CREATE FUNCTION test.hello\G
***** 1. row *****
      Function: hello
      sql_mode:
  Create Function: CREATE FUNCTION `test`.`hello`(s CHAR(20)) »
                   RETURNS CHAR(50)
                   RETURN CONCAT('Hello, ',s, '!')
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci

```

`character_set_client` is the session value of the `character_set_client` system variable when the routine was created. `collation_connection` is the session value of the `collation_connection` system variable when the routine was created. `Database Collation` is the collation of the database with which the routine is associated. These columns were added in MySQL 5.1.21.

12.5.4.9. SHOW CREATE TABLE Syntax

```
SHOW CREATE TABLE tbl_name
```

Shows the `CREATE TABLE` statement that creates the given table. This statement also works with views.

```

mysql> SHOW CREATE TABLE t\G
***** 1. row *****
      Table: t
  Create Table: CREATE TABLE t (
    id INT(11) default NULL auto_increment,
    s char(60) default NULL,
    PRIMARY KEY (id)
  ) ENGINE=MyISAM

```

`SHOW CREATE TABLE` quotes table and column names according to the value of the `SQL_QUOTE_SHOW_CREATE` option. See [Section 12.5.3, “SET Syntax”](#).

12.5.4.10. SHOW CREATE TRIGGER Syntax

```
SHOW CREATE TRIGGER trigger_name
```

This statement shows a `CREATE TRIGGER` statement that creates the given trigger.


```
mysql> SHOW CREATE TRIGGER ins_sum\G
***** 1. row *****
Trigger: ins_sum
sql_mode:
SQL Original Statement: CREATE DEFINER=`bob`@`localhost` TRIGGER ins_sum
                        BEFORE INSERT ON account
                        FOR EACH ROW SET @sum = @sum + NEW.amount
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
```

This statement was added in MySQL 5.1.21.

You can also obtain information about trigger objects from `INFORMATION_SCHEMA`, which contains a `TRIGGERS` table. See [Section 23.16, “The INFORMATION_SCHEMA TRIGGERS Table”](#).

12.5.4.11. SHOW CREATE VIEW Syntax

```
SHOW CREATE VIEW view_name
```

This statement shows a `CREATE VIEW` statement that creates the given view.

```
mysql> SHOW CREATE VIEW v\G
***** 1. row *****
View: v
Create View: CREATE ALGORITHM=UNDEFINED
              DEFINER=`bob`@`localhost`
              SQL SECURITY DEFINER VIEW
              `v` AS select 1 AS `a`,2 AS `b`
character_set_client: latin1
collation_connection: latin1_swedish_ci
```

`character_set_client` is the session value of the `character_set_client` system variable when the routine was created. `collation_connection` is the session value of the `collation_connection` system variable when the routine was created. These columns were added in MySQL 5.1.21.

Use of `SHOW CREATE VIEW` requires the `SHOW VIEW` privilege and the `SELECT` privilege for the view in question.

You can also obtain information about view objects from `INFORMATION_SCHEMA`, which contains a `VIEWS` table. See [Section 23.15, “The INFORMATION_SCHEMA VIEWS Table”](#).

12.5.4.12. SHOW DATABASES Syntax

```
SHOW {DATABASES | SCHEMAS}
[LIKE 'pattern' | WHERE expr]
```

`SHOW DATABASES` lists the databases on the MySQL server host. `SHOW SCHEMAS` is a synonym for `SHOW DATABASES`. The `LIKE` clause, if present, indicates which database names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#).

You see only those databases for which you have some kind of privilege, unless you have the global `SHOW DATABASES` privilege. You can also get this list using the `mysqlshow` command.

If the server was started with the `--skip-show-database` option, you cannot use this statement at all unless you have the `SHOW DATABASES` privilege.

`SHOW SCHEMAS` can also be used.

12.5.4.13. SHOW ENGINE Syntax

```
SHOW ENGINE engine_name {STATUS | MUTEX}
```

`SHOW ENGINE` displays operational information about a storage engine. The following statements currently are supported:

```
SHOW ENGINE INNODB STATUS
SHOW ENGINE INNODB MUTEX
SHOW ENGINE {NDB | NDBCLUSTER} STATUS
```

Older (and now deprecated) synonyms are `SHOW INNODB STATUS` for `SHOW ENGINE INNODB STATUS` and `SHOW MUTEX STATUS` for `SHOW ENGINE INNODB MUTEX`.

In MySQL 5.0, `SHOW ENGINE INNODB MUTEX` is invoked as `SHOW MUTEX STATUS`. The latter statement displays similar information but in a somewhat different output format.

`SHOW ENGINE BDB LOGS` formerly displayed status information about `BDB` log files. As of MySQL 5.1.12, the `BDB` storage engine is not supported, and this statement produces a warning.

`SHOW ENGINE INNODB STATUS` displays extensive information about the state of the `InnoDB` storage engine.

The `InnoDB` Monitors provide additional information about `InnoDB` processing. See [Section 13.5.11.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”](#).

`SHOW ENGINE INNODB MUTEX` displays `InnoDB` mutex statistics. From MySQL 5.1.2 to 5.1.14, the output fields are:

- **Type**

Always `InnoDB`.

- **Name**

The mutex name and the source file where it is implemented. Example: `&pool->mutex:mem0pool.c`

The mutex name indicates its purpose. For example, the `log_sys` mutex is used by the `InnoDB` logging subsystem and indicates how intensive logging activity is. The `buf_pool` mutex protects the `InnoDB` buffer pool.

- **Status**

The mutex status. The fields contains several values:

- `count` indicates how many times the mutex was requested.
- `spin_waits` indicates how many times the spinlock had to run.
- `spin_rounds` indicates the number of spinlock rounds. (`spin_rounds` divided by `spin_waits` provides the average round count.)
- `os_waits` indicates the number of operating system waits. This occurs when the spinlock did not work (the mutex was not locked during the spinlock and it was necessary to yield to the operating system and wait).
- `os_yields` indicates the number of times a the thread trying to lock a mutex gave up its timeslice and yielded to the operating system (on the presumption that allowing other threads to run will free the mutex so that it can be locked).
- `os_wait_times` indicates the amount of time (in ms) spent in operating system waits, if the `timed_mutexes` system variable is 1 (`ON`). If `timed_mutexes` is 0 (`OFF`), timing is disabled, so `os_wait_times` is 0. `timed_mutexes` is off by default.

From MySQL 5.1.15 on, the output fields are:

- **Type**

Always `InnoDB`.

- **Name**

The source file where the mutex is implemented, and the line number in the file where the mutex is created.

- **Status**

The `os_waits` value, which indicates the number of operating system waits. This occurs when the spinlock did not work (the mutex was not locked during the spinlock and it was necessary to yield to the operating system and wait).

Information from this statement can be used to diagnose system problems. For example, large values of `spin_waits` and `spin_rounds` may indicate scalability problems.

If the server has the `NDBCLUSTER` storage engine enabled, `SHOW ENGINE NDB STATUS` displays cluster status information such as the number of connected data nodes, the cluster connectstring, and cluster binlog epochs, as well as counts of various Cluster API objects created by the MySQL Server when connected to the cluster. Sample output from this statement is shown here:

```
mysql> SHOW ENGINE NDB STATUS;
+-----+-----+-----+
| Type | Name | Status |
+-----+-----+-----+
| ndbcluster | connection | cluster_node_id=7,
connected_host=192.168.0.103, connected_port=1186, number_of_data_nodes=4,
number_of_ready_data_nodes=3, connect_count=0 |
| ndbcluster | NdbTransaction | created=6, free=0, sizeof=212 |
| ndbcluster | NdbOperation | created=8, free=8, sizeof=660 |
| ndbcluster | NdbIndexScanOperation | created=1, free=1, sizeof=744 |
| ndbcluster | NdbIndexOperation | created=0, free=0, sizeof=664 |
| ndbcluster | NdbRecAttr | created=1285, free=1285, sizeof=60 |
| ndbcluster | NdbApiSignal | created=16, free=16, sizeof=136 |
| ndbcluster | NdbLabel | created=0, free=0, sizeof=196 |
| ndbcluster | NdbBranch | created=0, free=0, sizeof=24 |
| ndbcluster | NdbSubroutine | created=0, free=0, sizeof=68 |
| ndbcluster | NdbCall | created=0, free=0, sizeof=16 |
| ndbcluster | NdbBlob | created=1, free=1, sizeof=264 |
| ndbcluster | NdbReceiver | created=4, free=0, sizeof=68 |
| ndbcluster | binlog | latest_epoch=155467, latest_trans_epoch=148126,
latest_received_binlog_epoch=0, latest_handled_binlog_epoch=0,
latest_applied_binlog_epoch=0 |
+-----+-----+-----+
```

The rows with `connection` and `binlog` in the `Name` column were added to the output of this statement in MySQL 5.1. The `Status` column in each of these rows provides information about the MySQL server's connection to the cluster and about the cluster binary log's status, respectively. The `Status` information is in the form of comma-delimited set of name/value pairs.

The `connection` row's `Status` column contains the name/value pairs described in the following table:

Name	Value
<code>cluster_node_id</code>	The node ID of the MySQL server in the cluster
<code>connected_host</code>	The hostname or IP address of the cluster management server to which the MySQL server is connected
<code>connected_port</code>	The port used by the MySQL server to connect to the management server (<code>connected_host</code>)
<code>number_of_data_nodes</code>	The number of data nodes configured for the cluster (that is, the number of <code>[ndbd]</code> sections in the cluster <code>config.ini</code> file)
<code>number_of_ready_data_nodes</code>	The number of data nodes in the cluster that are actually running
<code>connect_count</code>	The number of active connections to cluster data nodes

The `binlog` row's `Status` column contains information relating to MySQL Cluster Replication. The name/value pairs it contains are described in the following table:

Name	Value
<code>latest_epoch</code>	The most recent epoch most recently run on this MySQL server (that is, the sequence number of the most recent transaction run on the server)
<code>latest_trans_epoch</code>	The most recent epoch processed by the cluster's data nodes
<code>latest_received_binlog_epoch</code>	The most recent epoch received by the binlog thread
<code>latest_handled_binlog_epoch</code>	The most recent epoch processed by the binlog thread (for writing to the binlog)
<code>latest_applied_binlog_epoch</code>	The most recent epoch actually written to the binlog

See [Section 16.10, “MySQL Cluster Replication”](#), for more information.

The remaining rows from the output of `SHOW ENGINE NDB STATUS` which are most likely to prove useful in monitoring the cluster are listed here by `Name`:

- **NdbTransaction**: The number and size of **NdbTransaction** objects that have been created. An **NdbTransaction** is created each time a table schema operation (such as **CREATE TABLE** or **ALTER TABLE**) is performed on an **NDB** table.
- **NdbOperation**: The number and size of **NdbOperation** objects that have been created.
- **NdbIndexScanOperation**: The number and size of **NdbIndexScanOperation** objects that have been created.
- **NdbIndexOperation**: The number and size of **NdbIndexOperation** objects that have been created.
- **NdbRecAttr**: The number and size of **NdbRecAttr** objects that have been created. In general, one of these is created each time a data manipulation statement is performed by an SQL node.
- **NdbBlob**: The number and size of **NdbBlob** objects that have been created. An **NdbBlob** is created for each new operation involving a **BLOB** column in an **NDB** table.
- **NdbReceiver**: The number and size of any **NdbReceiver** object that have been created. The number in the **created** column is the same as the number of data nodes in the cluster to which the MySQL server has connected.

Note

SHOW ENGINE NDB STATUS returns an empty result if no operations involving **NDB** tables have been performed during the current session by the MySQL client accessing the SQL node on which this statement is run.

MySQL Enterprise

The **SHOW ENGINE engine_name STATUS** statement provides valuable information about the state of your server. For expert interpretation of this information, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

12.5.4.14. SHOW ENGINES Syntax

SHOW [STORAGE] ENGINES

SHOW ENGINES displays status information about the server's storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is. **SHOW TABLE TYPES** is a deprecated synonym.

```
mysql> SHOW ENGINES\G
***** 1. row *****
Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
XA: NO
Savepoints: NO
***** 2. row *****
Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
Transactions: NO
XA: NO
Savepoints: NO
***** 3. row *****
Engine: InnoDB
Support: YES
Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
XA: YES
Savepoints: YES
***** 4. row *****
Engine: EXAMPLE
Support: YES
Comment: Example storage engine
Transactions: NO
XA: NO
Savepoints: NO
***** 5. row *****
Engine: ARCHIVE
Support: YES
Comment: Archive storage engine
Transactions: NO
XA: NO
Savepoints: NO
***** 6. row *****
Engine: CSV
Support: YES
Comment: CSV storage engine
```

```

Transactions: NO
XA: NO
Savepoints: NO
***** 7. row *****
Engine: BLACKHOLE
Support: YES
Comment: /dev/null storage engine (anything you write »
to it disappears)
Transactions: NO
XA: NO
Savepoints: NO
***** 8. row *****
Engine: FEDERATED
Support: YES
Comment: Federated MySQL storage engine
Transactions: NO
XA: NO
Savepoints: NO
***** 9. row *****
Engine: MRG_MYISAM
Support: YES
Comment: Collection of identical MyISAM tables
Transactions: NO
XA: NO
Savepoints: NO

```

The output from `SHOW ENGINES` may vary according to the MySQL version used and other factors. The values shown in the `Support` column indicate the server's level of support for different features, as shown here:

Value	Meaning
YES	The feature is supported and is active.
NO	The feature is not supported.
DISABLED	The feature is supported but has been disabled.

A value of `NO` means that the server was compiled without support for the feature, so it cannot be activated at runtime.

A value of `DISABLED` occurs either because the server was started with an option that disables the feature, or because not all options required to enable it were given. In the latter case, the error log file should contain a reason indicating why the option is disabled. See [Section 5.11.2, “The Error Log”](#).

You might also see `DISABLED` for a storage engine if the server was compiled to support it, but was started with a `--skip-engine` option. Only the `InnoDB`, and `MERGE` engines can be disabled in this way. For the `NDB Cluster` storage engine, `DISABLED` means the server was compiled with support for MySQL Cluster, but was not started with the `--ndb-cluster` option.

All MySQL servers support `MyISAM` tables, because `MyISAM` is the default storage engine.

The `Transactions`, `XA`, and `Savepoints` columns were added in MySQL 5.1.2. They indicate whether the storage engine supports transactions, XA transactions, and savepoints, respectively.

12.5.4.15. SHOW ERRORS Syntax

```

SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS

```

This statement is similar to `SHOW WARNINGS`, except that instead of displaying errors, warnings, and notes, it displays only errors.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See [Section 12.2.7, “SELECT Syntax”](#).

The `SHOW COUNT(*) ERRORS` statement displays the number of errors. You can also retrieve this number from the `error_count` variable:

```

SHOW COUNT(*) ERRORS;
SELECT @@error_count;

```

For more information, see [Section 12.5.4.32, “SHOW WARNINGS Syntax”](#).

12.5.4.16. SHOW EVENTS

```

SHOW EVENTS [FROM schema_name]

```

```
[LIKE 'pattern' | WHERE expr]
```

In its simplest form, **SHOW EVENTS** lists all of the events in the current schema:

```
mysql> SELECT CURRENT_USER(), SCHEMA();
+-----+-----+
| CURRENT_USER() | SCHEMA() |
+-----+-----+
| jon@ghidora    | myschema |
+-----+-----+
1 row in set (0.00 sec)

mysql> SHOW EVENTS\G
***** 1. row *****
      Db: myschema
      Name: e_daily
      Definer: jon@ghidora
      Time zone: SYSTEM
      Type: RECURRING
      Execute at: NULL
      Interval value: 10
      Interval field: INTERVAL_SECOND
      Starts: 2006-02-09 10:41:23
      Ends: 0000-00-00 00:00:00
      Status: ENABLED
      Originator: 0
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
```

The **LIKE** clause, if present, indicates which event names to match. The **WHERE** clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#).

The columns in the output of **SHOW EVENTS** — which are similar to, but not identical to the columns in the **INFORMATION_SCHEMA.EVENTS** table — are shown here:

- **Db**: The schema (database) on which the event is defined.
- **Name**: The name of the event.
- **Time zone**: The time zone in effect when schedule for the event was last modified. If the event's schedule has not been modified since the event was created, then this is the time zone that was in effect at the event's creation. The default value is **SYSTEM**.

This column was added in MySQL 5.1.17. See [Section C.1.10, “Changes in release 5.1.17 \(04 April 2007\)”](#), for important information if you are using the Event Scheduler and are upgrading from MySQL 5.1.16 (or earlier) to MySQL 5.1.17 (or later).

- **Definer**: The user account (*username@hostname*) which created the event.
- **Type**: One of the two values **ONE TIME** (transient) or **RECURRING**.
- **Execute At**: The date and time when a transient event is set to execute. Shown as a **DATETIME** value.

For a recurring event, the value of this column is always **NULL**.

- **Interval Value**: For a recurring event, the number of intervals to wait between event executions.

For a transient event, the value of this column is always **NULL**.

- **Interval Field**: The time units used for the interval which a recurring event waits before repeating.

For a transient event, the value of this column is always **NULL**.

- **Starts**: The start date and time for a recurring event. This is displayed as a **DATETIME** value, and is empty if no start date and time are defined for the event. (Prior to MySQL 5.1.8, it defaulted to '0000-00-00 00:00:00' in such cases.)

For a transient event, the value of this column is always **NULL**.

- **Ends**: The end date and time for a recurring event. This is displayed as a **DATETIME** value, and defaults to '0000-00-00 00:00:00' if no end date and time is defined for the event.

For a transient event, the value of this column is always **NULL**.

- **Status:** The event status. One of `ENABLED`, `DISABLED`, or `SLAVESIDE_DISABLED`.
`SLAVESIDE_DISABLED` was added in MySQL 5.1.18. This value indicates that the creation of the event occurred on another MySQL server acting as a replication master and replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave.
- **Originator:** The server ID of the MySQL server on which the event was created. Defaults to 0. This column was added in MySQL 5.1.18.
- **character_set_client** is the session value of the `character_set_client` system variable when the routine was created. **collation_connection** is the session value of the `collation_connection` system variable when the routine was created. **Database Collation** is the collation of the database with which the routine is associated. These columns were added in MySQL 5.1.21.

For more information about `SLAVE_DISABLED` and the **Originator** column, see [Section 15.3.1.5, “Replication of Invoked Features”](#).

Note that the action statement is not shown in the output of `SHOW EVENTS`.

Prior to MySQL 5.1.17, the values displayed for **Starts** and **Ends** (other than `'0000-00-00 00:00:00'`) were shown using Universal Time ([Bug#16420](#)). Beginning with MySQL 5.1.17, these times are all given in terms of local time as determined by the MySQL server's `time_zone` setting. See also [Section 23.20, “The INFORMATION_SCHEMA EVENTS Table”](#).

To see events for a different schema, you can use the **FROM** clause. For example, if the `test` schema had been selected in the preceding example, you could view events defined on `myschema` using the following statement:

```
SHOW EVENTS FROM myschema;
```

You can filter the list returned by this statement on the event name using **LIKE** plus a pattern.

This statement was added in MySQL 5.1.6.

See also [Section 23.20, “The INFORMATION_SCHEMA EVENTS Table”](#).

Note

In MySQL 5.1.11 and earlier, `SHOW EVENTS` displayed only those events for which the current user was the definer, and the `SHOW FULL EVENTS` statement was used for viewing events defined by all users on a given schema. `SHOW FULL EVENTS` was removed in MySQL 5.1.12.

12.5.4.17. SHOW GRANTS Syntax

```
SHOW GRANTS [FOR user]
```

This statement lists the **GRANT** statement or statements that must be issued to duplicate the privileges that are granted to a MySQL user account. The account is named using the same format as for the **GRANT** statement; for example, `'jeffrey'@'localhost'`. If you specify only the username part of the account name, a hostname part of `'%'` is used. For additional information about specifying account names, see [Section 12.5.1.3, “GRANT Syntax”](#).

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+-----+-----+
| Grants for root@localhost |
+-----+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+-----+-----+
```

To list the privileges granted to the account that you are using to connect to the server, you can use any of the following statements:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

As of MySQL 5.1.12, if `SHOW GRANTS FOR CURRENT_USER` (or any of the equivalent syntaxes) is used in **DEFINER** context, such as within a stored procedure that is defined with **SQL SECURITY DEFINER**, the grants displayed are those of the definer and

not the invoker.

SHOW GRANTS displays only the privileges granted explicitly to the named account. Other privileges might be available to the account, but they are not displayed. For example, if an anonymous account exists, the named account might be able to use its privileges, but **SHOW GRANTS** will not display them.

12.5.4.18. SHOW INDEX Syntax

```
SHOW INDEX FROM tbl_name [FROM db_name]
```

SHOW INDEX returns table index information. The format resembles that of the [SQLStatistics](#) call in ODBC.

SHOW INDEX returns the following fields:

- **Table**
The name of the table.
- **Non_unique**
0 if the index cannot contain duplicates, 1 if it can.
- **Key_name**
The name of the index.
- **Seq_in_index**
The column sequence number in the index, starting with 1.
- **Column_name**
The column name.
- **Collation**
How the column is sorted in the index. In MySQL, this can have values 'A' (Ascending) or **NULL** (Not sorted).
- **Cardinality**
An estimate of the number of unique values in the index. This is updated by running [ANALYZE TABLE](#) or [myisamchk -a](#). **Cardinality** is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.
- **Sub_part**
The number of indexed characters if the column is only partly indexed, **NULL** if the entire column is indexed.
- **Packed**
Indicates how the key is packed. **NULL** if it is not.
- **Null**
Contains **YES** if the column may contain **NULL**. If not, the column contains **NO**.
- **Index_type**
The index method used ([BTREE](#), [FULLTEXT](#), [HASH](#), [RTREE](#)).
- **Comment**
Various remarks.

You can use *db_name.tbl_name* as an alternative to the *tbl_name FROM db_name* syntax. These two statements are equival-

ent:

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

SHOW KEYS is a synonym for **SHOW INDEX**. You can also list a table's indexes with the `mysqlshow -k db_name tbl_name` command.

12.5.4.19. SHOW INNODB STATUS Syntax

```
SHOW INNODB STATUS
```

In MySQL 5.1, this is a deprecated synonym for **SHOW ENGINE INNODB STATUS**. See [Section 12.5.4.13, “SHOW ENGINE Syntax”](#).

12.5.4.20. SHOW OPEN TABLES Syntax

```
SHOW OPEN TABLES [FROM db_name]
[LIKE 'pattern' | WHERE expr]
```

SHOW OPEN TABLES lists the non-**TEMPORARY** tables that are currently open in the table cache. See [Section 6.4.8, “How MySQL Opens and Closes Tables”](#). The **LIKE** clause, if present, indicates which table names to match. The **WHERE** clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#).

SHOW OPEN TABLES returns the following fields:

- **Database**

The database containing the table.

- **Table**

The table name.

- **In_use**

The number of table locks or lock requests there are for the table. For example, if one client acquires a lock for a table using **LOCK TABLE t1 WRITE**, **In_use** will be 1. If another client issues **LOCK TABLE t1 WRITE** while the table remains locked, the client will block waiting for the lock, but the lock request causes **In_use** to be 2. If the count is zero, the table is open but not currently being used.

- **Name_locked**

Whether the table name is locked. Name locking is used for operations such as dropping or renaming tables.

12.5.4.21. SHOW PLUGINS Syntax

```
SHOW PLUGINS
```

SHOW PLUGINS displays information about known plugins.

```
mysql> SHOW PLUGINS;
+-----+-----+-----+-----+
| Name      | Status | Type          | Library |
+-----+-----+-----+-----+
| MEMORY    | ACTIVE | STORAGE ENGINE | NULL    |
| MyISAM    | ACTIVE | STORAGE ENGINE | NULL    |
| InnoDB    | ACTIVE | STORAGE ENGINE | NULL    |
| ARCHIVE   | ACTIVE | STORAGE ENGINE | NULL    |
| CSV       | ACTIVE | STORAGE ENGINE | NULL    |
| BLACKHOLE | ACTIVE | STORAGE ENGINE | NULL    |
| FEDERATED | ACTIVE | STORAGE ENGINE | NULL    |
| MRG_MYISAM | ACTIVE | STORAGE ENGINE | NULL    |
+-----+-----+-----+-----+
```

SHOW PLUGIN was added in MySQL 5.1.5 and renamed to **SHOW PLUGINS** in 5.1.9. (As of 5.1.9, **SHOW PLUGIN** is deprecated and

generates a warning.)

12.5.4.22. SHOW PRIVILEGES Syntax

SHOW PRIVILEGES

SHOW PRIVILEGES shows the list of system privileges that the MySQL server supports. The exact list of privileges depends on the version of your server.

```
mysql> SHOW PRIVILEGES\G
***** 1. row *****
Privilege: Alter
Context: Tables
Comment: To alter the table
***** 2. row *****
Privilege: Alter routine
Context: Functions,Procedures
Comment: To alter or drop stored functions/procedures
***** 3. row *****
Privilege: Create
Context: Databases,Tables,Indexes
Comment: To create new databases and tables
***** 4. row *****
Privilege: Create routine
Context: Functions,Procedures
Comment: To use CREATE FUNCTION/PROCEDURE
***** 5. row *****
Privilege: Create temporary tables
Context: Databases
Comment: To use CREATE TEMPORARY TABLE
...
```

12.5.4.23. SHOW PROCEDURE CODE and SHOW FUNCTION CODE Syntax

SHOW {PROCEDURE | FUNCTION} CODE *sp_name*

These statements are MySQL extensions that are available only for servers that have been built with debugging support. They display a representation of the internal implementation of the named routine. The statements require that you be the owner of the routine or have **SELECT** access to the `mysql.proc` table.

If the named routine is available, each statement produces a result set. Each row in the result set corresponds to one “instruction” in the routine. The first column is **Pos**, which is an ordinal number beginning with 0. The second column is **Instruction**, which contains an SQL statement (usually changed from the original source), or a directive which has meaning only to the stored-routine handler.

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE p1 ()
-> BEGIN
->   DECLARE fanta INT DEFAULT 55;
->   DROP TABLE t2;
->   LOOP
->     INSERT INTO t3 VALUES (fanta);
->   END LOOP;
-> END//
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW PROCEDURE CODE p1//
+-----+-----+
| Pos | Instruction |
+-----+-----+
| 0   | set fanta@0 55 |
| 1   | stmt 9 "DROP TABLE t2" |
| 2   | stmt 5 "INSERT INTO t3 VALUES (fanta)" |
| 3   | jump 2 |
+-----+-----+
4 rows in set (0.00 sec)
```

In this example, the non-executable **BEGIN** and **END** statements have disappeared, and for the **DECLARE variable_name** statement, only the executable part appears (the part where the default is assigned). For each statement that is taken from source, there is a code word **stmt** followed by a type (9 means **DROP**, 5 means **INSERT**, and so on). The final row contains an instruction **jump 2**, meaning **GOTO instruction #2**.

These statements were added in MySQL 5.1.3.

12.5.4.24. SHOW PROCEDURE STATUS and SHOW FUNCTION STATUS Syntax

```
SHOW {PROCEDURE | FUNCTION} STATUS
[LIKE 'pattern' | WHERE expr]
```

These statements are MySQL extensions. They return characteristics of routines, such as the database, name, type, creator, creation and modification dates, and character set information. The **LIKE** clause, if present, indicates which procedure or function names to match. The **WHERE** clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#).

```
mysql> SHOW FUNCTION STATUS LIKE 'hello'\G
***** 1. row *****
      Db: test
      Name: hello
      Type: FUNCTION
      Definer: testuser@localhost
      Modified: 2004-08-03 15:29:37
      Created: 2004-08-03 15:29:37
      Security_type: DEFINER
      Comment:
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
```

character_set_client is the session value of the **character_set_client** system variable when the routine was created. **collation_connection** is the session value of the **collation_connection** system variable when the routine was created. **Database Collation** is the collation of the database with which the routine is associated. These columns were added in MySQL 5.1.21.

You can also get information about stored routines from the **ROUTINES** table in **INFORMATION_SCHEMA**. See [Section 23.14, “The INFORMATION_SCHEMA ROUTINES Table”](#).

12.5.4.25. SHOW PROCESSLIST Syntax

```
SHOW [FULL] PROCESSLIST
```

SHOW PROCESSLIST shows you which threads are running. You can also get this information from the **INFORMATION_SCHEMA PROCESSLIST** table or the **mysqladmin processlist** command. If you have the **PROCESS** privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MySQL account that you are using). If you do not use the **FULL** keyword, only the first 100 characters of each statement are shown in the **Info** field.

MySQL Enterprise

Subscribers to MySQL Enterprise Monitor receive instant notification and expert advice on resolution when there are too many concurrent processes. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

This statement is very useful if you get the “too many connections” error message and want to find out what is going on. MySQL reserves one extra connection to be used by accounts that have the **SUPER** privilege, to ensure that administrators should always be able to connect and check the system (assuming that you are not giving this privilege to all your users).

Threads can be killed with the **KILL** statement. See [Section 12.5.5.3, “KILL Syntax”](#).

Here is an example of what **SHOW PROCESSLIST** output looks like:

```
mysql> SHOW FULL PROCESSLIST\G
***** 1. row *****
Id: 1
User: system user
Host:
db: NULL
Command: Connect
Time: 1030455
State: Waiting for master to send event
Info: NULL
***** 2. row *****
Id: 2
User: system user
Host:
db: NULL
Command: Connect
Time: 1004
State: Has read all relay log; waiting for the slave
      I/O thread to update it
Info: NULL
***** 3. row *****
```

```
Id: 3112
User: replikator
Host: artemis:2204
db: NULL
Command: Binlog Dump
Time: 2144
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
***** 4. row *****
Id: 3113
User: replikator
Host: iconnect2:45781
db: NULL
Command: Binlog Dump
Time: 2086
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
***** 5. row *****
Id: 3123
User: stefan
Host: localhost
db: apollon
Command: Query
Time: 0
State: NULL
Info: SHOW FULL PROCESSLIST
5 rows in set (0.00 sec)
```

The columns have the following meaning:

- **Id**

The connection identifier.

- **User**

The MySQL user who issued the statement. If this is **system user**, it refers to a non-client thread spawned by the server to handle tasks internally. This could be the I/O or SQL thread used on replication slaves or a delayed-row handler. **unauthenticated user** refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet been done. **event_scheduler** refers to the thread that monitors scheduled events. For **system user** or **event_scheduler**, there is no host specified in the **Host** column.

- **Host**

The hostname of the client issuing the statement (except for **system user** where there is no host). **SHOW PROCESSLIST** reports the hostname for TCP/IP connections in **host_name:client_port** format to make it easier to determine which client is doing what.

- **db**

The default database, if one is selected, otherwise **NULL**.

- **Command**

The type of command the thread is executing. Descriptions for thread commands can be found at [Section 6.5.5, “Examining Thread Information”](#). The value of this column corresponds to the **COM_XXX** commands of the client/server protocol. See [Section 5.2.5, “Status Variables”](#)

- **Time**

The time in seconds that the thread has been in its current state.

- **State**

An action, event, or state that indicates what the thread is doing. Descriptions for **State** values can be found at [Section 6.5.5, “Examining Thread Information”](#).

Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

For the **SHOW PROCESSLIST** statement, the value of **State** is **NULL**.

- **Info**

The statement that the thread is executing, or **NULL** if it is not executing any statement.

12.5.4.26. SHOW SCHEDULER STATUS Syntax

```
SHOW SCHEDULER STATUS
```

This statement provides debugging information regarding the Event Scheduler's state. It is supported only in **-debug** builds of MySQL 5.1.11, and was removed in 5.1.12 and subsequent releases.

Sample output is shown here:

Name	Value
scheduler state	INITIALIZED
thread_id	NULL
scheduler last locked at	init_scheduler::313
scheduler last unlocked at	init_scheduler::318
scheduler waiting on condition	0
scheduler workers count	0
scheduler executed events	0
scheduler data locked	0
queue element count	1
queue data locked	0
queue data attempting lock	0
queue last locked at	create_event::218
queue last unlocked at	create_event::222
queue last attempted lock at	::0
queue waiting on condition	0
next activation at	0-00-00 00:00:00

In MySQL 5.1.12 and later, this information can be obtained using `mysqladmin debug`. (See [Section 7.10, “mysqladmin — Client for Administering a MySQL Server”](#).) For more information about obtaining Event Scheduler status information, see [Section 21.4, “Event Scheduler Status”](#).

12.5.4.27. SHOW STATUS Syntax

```
SHOW [GLOBAL | SESSION] STATUS
[LIKE 'pattern' | WHERE expr]
```

SHOW STATUS provides server status information. This information also can be obtained using the `mysqladmin extended-status` command. The **LIKE** clause, if present, indicates which variable names to match. The **WHERE** clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#).

Partial output is shown here. The list of names and values may be different for your server. The meaning of each variable is given in [Section 5.2.5, “Status Variables”](#).

```
mysql> SHOW STATUS;
```

Variable_name	Value
Aborted_clients	0
Aborted_connects	0
Bytes_received	155372598
Bytes_sent	1176560426
Connections	30023
Created_tmp_disk_tables	0
Created_tmp_tables	8340
Created_tmp_files	60
...	
Open_tables	1
Open_files	2
Open_streams	0
Opened_tables	44600
Questions	2026873
...	
Table_locks_immediate	1920382
Table_locks_waited	0
Threads_cached	0
Threads_created	30022
Threads_connected	1
Threads_running	1

Uptime	80380

With a **LIKE** clause, the statement displays only rows for those variables with names that match the pattern:

```
mysql> SHOW STATUS LIKE 'Key%';
```

Variable_name	Value
Key_blocks_used	14955
Key_read_requests	96854827
Key_reads	162040
Key_write_requests	7589728
Key_writes	3813196

With the **GLOBAL** modifier, **SHOW STATUS** displays the status values for all connections to MySQL. With **SESSION**, it displays the status values for the current connection. If no modifier is present, the default is **SESSION**. **LOCAL** is a synonym for **SESSION**.

Some status variables have only a global value. For these, you get the same value for both **GLOBAL** and **SESSION**. The scope for each status variable is listed at [Section 5.2.5, “Status Variables”](#).

MySQL Enterprise

Status variables provide valuable clues to the state of your servers. For expert interpretation of the information provided by status variables, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

12.5.4.28. SHOW TABLE STATUS Syntax

```
SHOW TABLE STATUS [FROM db_name]
[LIKE 'pattern' | WHERE expr]
```

SHOW TABLE STATUS works like **SHOW TABLES**, but provides a lot of information about each table. You can also get this list using the `mysqlshow --status db_name` command. The **LIKE** clause, if present, indicates which table names to match. The **WHERE** clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#).

This statement also displays information about views.

SHOW TABLE STATUS returns the following fields:

- **Name**
The name of the table.
- **Engine**
The storage engine for the table. See [Chapter 13, Storage Engines](#).
- **Version**
The version number of the table's `.frm` file.
- **Row_format**
The row storage format (**Fixed**, **Dynamic**, **Compressed**, **Redundant**, **Compact**). The format of **InnoDB** tables is reported as **Redundant** or **Compact**.
- **Rows**
The number of rows. Some storage engines, such as **MyISAM**, store the exact count. For other storage engines, such as **InnoDB**, this value is an approximation, and may vary from the actual value by as much as 40 to 50%. In such cases, use **SELECT COUNT(*)** to obtain an accurate count.

The **Rows** value is **NULL** for tables in the **INFORMATION_SCHEMA** database.
- **Avg_row_length**

The average row length.

- **Data_length**

The length of the data file.

- **Max_data_length**

The maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

- **Index_length**

The length of the index file.

- **Data_free**

The number of allocated but unused bytes.

- **Auto_increment**

The next **AUTO_INCREMENT** value.

- **Create_time**

When the table was created.

- **Update_time**

When the data file was last updated. For some storage engines, this value is **NULL**. For example, **InnoDB** stores multiple tables in its tablespace and the data file timestamp does not apply.

- **Check_time**

When the table was last checked. Not all storage engines update this time, in which case the value is always **NULL**.

- **Collation**

The table's character set and collation.

- **Checksum**

The live checksum value (if any).

- **Create_options**

Extra options used with **CREATE TABLE**. The original options supplied when **CREATE TABLE** is called are retained and the options reported here may differ from the active table settings and options.

- **Comment**

The comment used when creating the table (or information as to why MySQL could not access the table information).

In the table comment, **InnoDB** tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table.

For **MEMORY** tables, the **Data_length**, **Max_data_length**, and **Index_length** values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.

For **NDB Cluster** tables, the output of this statement shows appropriate values for the **Avg_row_length** and **Data_length** columns, with the exception that **BLOB** columns are not taken into account. In addition, the number of replicas is shown in the **Comment** column (as **number_of_replicas**).

For views, all the fields displayed by **SHOW TABLE STATUS** are **NULL** except that **Name** indicates the view name and **Comment** says

view.

12.5.4.29. SHOW TABLES Syntax

```
SHOW [FULL] TABLES [FROM db_name]
      [LIKE 'pattern' | WHERE expr]
```

SHOW TABLES lists the non-**TEMPORARY** tables in a given database. You can also get this list using the `mysqlshow db_name` command. The **LIKE** clause, if present, indicates which table names to match. The **WHERE** clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#).

This statement also lists any views in the database. The **FULL** modifier is supported such that **SHOW FULL TABLES** displays a second output column. Values for the second column are **BASE TABLE** for a table and **VIEW** for a view.

Note

If you have no privileges for a table, the table does not show up in the output from **SHOW TABLES** or `mysqlshow db_name`.

12.5.4.30. SHOW TRIGGERS Syntax

```
SHOW TRIGGERS [FROM db_name]
      [LIKE 'pattern' | WHERE expr]
```

SHOW TRIGGERS lists the triggers currently defined on the MySQL server. This statement requires the **SUPER** privilege. The **LIKE** clause, if present, indicates which trigger names to match. The **WHERE** clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#).

For the trigger `ins_sum` as defined in [Section 20.3, “Using Triggers”](#), the output of this statement is as shown here:

```
mysql> SHOW TRIGGERS LIKE 'acc%\G
***** 1. row *****
      Trigger: ins_sum
      Event: INSERT
      Table: account
      Statement: SET @sum = @sum + NEW.amount
      Timing: BEFORE
      Created: NULL
      sql_mode:
      Definer: myname@localhost
character_set_client: latin1
collation_connection: latin1_swedish_ci
Database Collation: latin1_swedish_ci
```

character_set_client is the session value of the **character_set_client** system variable when the trigger was created. **collation_connection** is the session value of the **collation_connection** system variable when the trigger was created. **Database Collation** is the collation of the database with which the trigger is associated. These columns were added in MySQL 5.1.21.

Note

When using a **LIKE** clause with **SHOW TRIGGERS**, the expression to be matched (*expr*) is compared with the name of the table on which the trigger is declared, and not with the name of the trigger:

```
mysql> SHOW TRIGGERS LIKE 'ins%';
Empty set (0.01 sec)
```

A brief explanation of the columns in the output of this statement is shown here:

- **Trigger**

The name of the trigger.

- **Event**

The event that causes trigger activation: one of **'INSERT'**, **'UPDATE'**, or **'DELETE'**.

- **Table**

The table for which the trigger is defined.

- **Statement**

The statement to be executed when the trigger is activated. This is the same as the text shown in the `ACTION_STATEMENT` column of `INFORMATION_SCHEMA.TRIGGERS`.

- **Timing**

One of the two values `'BEFORE'` or `'AFTER'`.

- **Created**

Currently, the value of this column is always `NULL`.

- **sql_mode**

The SQL mode in effect when the trigger executes.

- **Definer**

The account that created the trigger.

See also [Section 23.16, “The INFORMATION_SCHEMA TRIGGERS Table”](#).

12.5.4.31. SHOW VARIABLES Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES
     [LIKE 'pattern' | WHERE expr]
```

`SHOW VARIABLES` shows the values of MySQL system variables. This information also can be obtained using the `mysqladmin variables` command. The `LIKE` clause, if present, indicates which variable names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in [Section 23.27, “Extensions to SHOW Statements”](#).

With the `GLOBAL` modifier, `SHOW VARIABLES` displays the values that are used for new connections to MySQL. With `SESSION`, it displays the values that are in effect for the current connection. If no modifier is present, the default is `SESSION`. `LOCAL` is a synonym for `SESSION`.

If the default system variable values are unsuitable, you can set them using command options when `mysqld` starts, and most can be changed at runtime with the `SET` statement. See [Section 5.2.4, “Using System Variables”](#), and [Section 12.5.3, “SET Syntax”](#).

Partial output is shown here. The list of names and values may be different for your server. [Section 5.2.3, “System Variables”](#), describes the meaning of each variable, and [Section 6.5.2, “Tuning Server Parameters”](#), provides information about tuning them.

```
mysql> SHOW VARIABLES;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
| automatic_sp_privileges | ON |
| back_log | 50 |
| basedir | /home/jon/bin/mysql-5.1/ |
| binlog_cache_size | 32768 |
| bulk_insert_buffer_size | 8388608 |
| character_set_client | latin1 |
| character_set_connection | latin1 |
| ... | ... |
| max_user_connections | 0 |
| max_write_lock_count | 4294967295 |
| multi_range_count | 256 |
| myisam_data_pointer_size | 6 |
| myisam_max_sort_file_size | 2147483647 |
| myisam_recover_options | OFF |
| myisam_repair_threads | 1 |
| myisam_sort_buffer_size | 8388608 |
| ndb_autoincrement_prefetch_sz | 32 |
| ndb_cache_check_time | 0 |
| ndb_force_send | ON |
| ... | ... |
```

time_zone	SYSTEM
timed_mutexes	OFF
tmp_table_size	33554432
tmpdir	
transaction_alloc_block_size	8192
transaction_prealloc_size	4096
tx_isolation	REPEATABLE-READ
updatable_views_with_limit	YES
version	5.1.6-alpha-log
version_comment	Source distribution
version_compile_machine	i686
version_compile_os	suse-linux
wait_timeout	28800

With a [LIKE](#) clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a [LIKE](#) clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the `'%'` wildcard character in a [LIKE](#) clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because `'_'` is a wildcard that matches any single character, you should escape it as `'\_'` to match it literally. In practice, this is rarely necessary.

12.5.4.32. SHOW WARNINGS Syntax

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

[SHOW WARNINGS](#) shows the error, warning, and note messages that resulted from the last statement that generated messages, or nothing if the last statement that used a table generated no messages. A related statement, [SHOW ERRORS](#), shows only the errors. See [Section 12.5.4.15, “SHOW ERRORS Syntax”](#).

The list of messages is reset for each new statement that uses a table.

The [SHOW COUNT\(*\) WARNINGS](#) statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the `warning_count` variable:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

The value of `warning_count` might be greater than the number of messages displayed by [SHOW WARNINGS](#) if the `max_error_count` system variable is set so low that not all messages are stored. An example shown later in this section demonstrates how this can happen.

The [LIMIT](#) clause has the same syntax as for the [SELECT](#) statement. See [Section 12.2.7, “SELECT Syntax”](#).

The MySQL server sends back the total number of errors, warnings, and notes resulting from the last statement. If you are using the C API, this value can be obtained by calling `mysql_warning_count()`. See [Section 25.2.3.72, “mysql_warning_count\(\)”](#).

Warnings are generated for statements such as [LOAD DATA INFILE](#) and DML statements such as [INSERT](#), [UPDATE](#), [CREATE TABLE](#), and [ALTER TABLE](#).

The following [DROP TABLE](#) statement results in a note:

```
mysql> DROP TABLE IF EXISTS no_such_table;
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Note  | 1051 | Unknown table 'no_such_table'             |
+-----+-----+-----+
```

Here is a simple example that shows a syntax warning for [CREATE TABLE](#) and conversion warnings for [INSERT](#):

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4)) TYPE=MyISAM;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1287
Message: 'TYPE=storage_engine' is deprecated, use
        'ENGINE=storage_engine' instead
1 row in set (0.00 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'),(NULL,'test'),
-> (300,'Open Source');
Query OK, 3 rows affected, 4 warnings (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 4

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 1
***** 2. row *****
Level: Warning
Code: 1263
Message: Data truncated, NULL supplied to NOT NULL column 'a' at row 2
***** 3. row *****
Level: Warning
Code: 1264
Message: Data truncated, out of range for column 'a' at row 3
***** 4. row *****
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 3
4 rows in set (0.00 sec)
```

The maximum number of error, warning, and note messages to store is controlled by the `max_error_count` system variable. By default, its value is 64. To change the number of messages you want stored, change the value of `max_error_count`. In the following example, the `ALTER TABLE` statement produces three warning messages, but only one is stored because `max_error_count` has been set to 1:

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_error_count | 64    |
+-----+-----+
1 row in set (0.00 sec)

mysql> SET max_error_count=1;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 3

mysql> SELECT @@warning_count;
+-----+
| @@warning_count |
+-----+
| 3               |
+-----+
1 row in set (0.01 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

To disable warnings, set `max_error_count` to 0. In this case, `warning_count` still indicates how many warnings have occurred, but none of the messages are stored.

You can set the `SQL_NOTES` session variable to 0 to cause `Note`-level warnings not to be recorded.

12.5.5. Other Administrative Statements

12.5.5.1. CACHE INDEX Syntax

```
CACHE INDEX
tbl_index_list [, tbl_index_list] ...
```

```
IN key_cache_name
tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name [, index_name] ...)]
```

The **CACHE INDEX** statement assigns table indexes to a specific key cache. It is used only for **MyISAM** tables.

The following statement assigns indexes from the tables **t1**, **t2**, and **t3** to the key cache named **hot_cache**:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
```

Table	Op	Msg_type	Msg_text
test.t1	assign_to_keycache	status	OK
test.t2	assign_to_keycache	status	OK
test.t3	assign_to_keycache	status	OK

The syntax of **CACHE INDEX** enables you to specify that only particular indexes from a table should be assigned to the cache. The current implementation assigns all the table's indexes to the cache, so there is no reason to specify anything other than the table name.

The key cache referred to in a **CACHE INDEX** statement can be created by setting its size with a parameter setting statement or in the server parameter settings. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Key cache parameters can be accessed as members of a structured system variable. See [Section 5.2.4.1, “Structured System Variables”](#).

A key cache must exist before you can assign indexes to it:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it become assigned to the default key cache again.

Index assignment affects the server globally: If one client assigns an index to a given cache, this cache is used for all queries involving the index, no matter which client issues the queries.

12.5.5.2. FLUSH Syntax

```
FLUSH [LOCAL | NO_WRITE_TO_BINLOG]
      flush_option [, flush_option] ...
```

The **FLUSH** statement clears or reloads various internal caches used by MySQL. To execute **FLUSH**, you must have the **RELOAD** privilege.

The **RESET** statement is similar to **FLUSH**. See [Section 12.5.5.5, “RESET Syntax”](#).

flush_option can be any of the following:

- **HOSTS**

Empties the host cache tables. You should flush the host tables if some of your hosts change IP number or if you get the error message **Host 'host_name' is blocked**. When more than **max_connect_errors** errors occur successively for a given host while connecting to the MySQL server, MySQL assumes that something is wrong and blocks the host from further connection requests. Flushing the host tables allows the host to attempt to connect again. See [Section B.1.2.5, “Host 'host_name' is blocked”](#). You can start **mysqld** with **--max_connect_errors=999999999** to avoid this error message.

- **DES_KEY_FILE**

Reloads the DES keys from the file that was specified with the **--des-key-file** option at server startup time.

- **LOGS**

Closes and reopens all log files. If binary logging is enabled, the sequence number of the binary log file is incremented by one relat-

ive to the previous file. On Unix, this is the same thing as sending a `SIGHUP` signal to the `mysqld` server (except on some Mac OS X 10.3 versions where `mysqld` ignores `SIGHUP` and `SIGQUIT`).

If the server was started with the `--log-error` option, `FLUSH LOGS` causes it to rename the current error log file with a suffix of `-old` and create a new empty log file. No renaming occurs if the `--log-error` option was not given.

- **MASTER (DEPRECATED)**. Deletes all binary logs, resets the binary log index file and creates a new binary log. `FLUSH MASTER` is deprecated in favor of `RESET MASTER`, and is supported for backwards compatibility only. See [Section 12.6.1.2, “RESET MASTER Syntax”](#).

- **PRIVILEGES**

Reloads the privileges from the grant tables in the `mysql` database.

- **QUERY CACHE**

Defragment the query cache to better utilize its memory. `FLUSH QUERY CACHE` does not remove any queries from the cache, unlike `RESET QUERY CACHE`.

- **SLAVE (DEPRECATED)**. Resets all replication slave parameters, including relay log files and replication position in the master's binary logs. `FLUSH SLAVE` is deprecated in favour of `RESET SLAVE`, and is supported for backwards compatibility only. See [Section 12.6.2.5, “RESET SLAVE Syntax”](#).

- **STATUS**

This option adds the current thread's session status variable values to the global values and resets the session values to zero. It also resets the counters for key caches (default and named) to zero and sets `Max_used_connections` to the current number of open connections. This is something you should use only when debugging a query. See [Section 1.8, “How to Report Bugs or Problems”](#).

- **{TABLE | TABLES} [tbl_name [, tbl_name] ...]**

When no tables are named, closes all open tables and forces all tables in use to be closed. This also flushes the query cache. With one or more table names, flushes only the given tables. `FLUSH TABLES` also removes all query results from the query cache, like the `RESET QUERY CACHE` statement.

- **TABLES WITH READ LOCK**

Closes all open tables and locks all tables for all databases with a read lock until you explicitly release the lock by executing `UNLOCK TABLES`. This is a very convenient way to get backups if you have a filesystem such as Veritas that can take snapshots in time.

`FLUSH TABLES WITH READ LOCK` acquires a global read lock and not table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits:

- `UNLOCK TABLES` commits a transaction only if any tables currently have been locked with `LOCK TABLES`. This does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table-level locks.
 - Beginning a transaction causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.
- **USER_RESOURCES**

Resets all per-hour user resources to zero. This enables clients that have reached their hourly connection, query, or update limits to resume activity immediately. `FLUSH USER_RESOURCES` does not apply to the limit on maximum simultaneous connections. See [Section 12.5.1.3, “GRANT Syntax”](#).

By default, `FLUSH` statements are written to the binary log so that such statements used on a MySQL server acting as a replication master will be replicated to replication slaves. Logging can be suppressed with the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

Note

`FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK` are not logged in any

- case because they would cause problems if replicated to a slave.

You can also access some of these statements with the `mysqladmin` utility, using the `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, or `flush-tables` commands.

Note

It is not possible in MySQL 5.1 to issue `FLUSH` statements within stored functions or triggers. However, you may use `FLUSH` in stored procedures, so long as these are not called from stored functions or triggers. See [Section D.1](#), “Restrictions on Stored Routines and Triggers”.

See also [Section 12.5.5.5](#), “RESET Syntax”, for information about how the `RESET` statement is used with replication.

12.5.5.3. KILL Syntax

```
KILL [CONNECTION | QUERY] thread_id
```

Each connection to `mysqld` runs in a separate thread. You can see which threads are running with the `SHOW PROCESSLIST` statement and kill a thread with the `KILL thread_id` statement.

`KILL` allows the optional `CONNECTION` or `QUERY` modifier:

- `KILL CONNECTION` is the same as `KILL` with no modifier: It terminates the connection associated with the given *thread_id*.
- `KILL QUERY` terminates the statement that the connection is currently executing, but leaves the connection itself intact.

If you have the `PROCESS` privilege, you can see all threads. If you have the `SUPER` privilege, you can kill all threads and statements. Otherwise, you can see and kill only your own threads and statements.

You can also use the `mysqladmin processlist` and `mysqladmin kill` commands to examine and kill threads.

Note

You cannot use `KILL` with the Embedded MySQL Server library, because the embedded server merely runs inside the threads of the host application. It does not create any connection threads of its own.

When you use `KILL`, a thread-specific kill flag is set for the thread. In most cases, it might take some time for the thread to die, because the kill flag is checked only at specific intervals:

- In `SELECT`, `ORDER BY` and `GROUP BY` loops, the flag is checked after reading a block of rows. If the kill flag is set, the statement is aborted.
- During `ALTER TABLE`, the kill flag is checked before each block of rows are read from the original table. If the kill flag was set, the statement is aborted and the temporary table is deleted.
- During `UPDATE` or `DELETE` operations, the kill flag is checked after each block read and after each updated or deleted row. If the kill flag is set, the statement is aborted. Note that if you are not using transactions, the changes are not rolled back.
- `GET_LOCK()` aborts and returns `NULL`.
- An `INSERT DELAYED` thread quickly flushes (inserts) all rows it has in memory and then terminates.
- If the thread is in the table lock handler (state: `Locked`), the table lock is quickly aborted.
- If the thread is waiting for free disk space in a write call, the write is aborted with a “disk full” error message.
-

Warning

Killing a `REPAIR TABLE` or `OPTIMIZE TABLE` operation on a `MyISAM` table results in a table that is corrupted and unusable. Any reads or writes to such a table fail until you optimize or repair it again (without interruption).

12.5.5.4. LOAD INDEX INTO CACHE Syntax

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...
tbl_index_list:
  tbl_name
  [[INDEX|KEY] (index_name[, index_name] ...)]
  [IGNORE LEAVES]
```

The **LOAD INDEX INTO CACHE** statement preloads a table index into the key cache to which it has been assigned by an explicit **CACHE INDEX** statement, or into the default key cache otherwise. **LOAD INDEX INTO CACHE** is used only for **MyISAM** tables. It is not supported for tables having user-defined partitioning (see [Section 17.5, “Restrictions and Limitations on Partitioning”](#)).

The **IGNORE LEAVES** modifier causes only blocks for the non-leaf nodes of the index to be preloaded.

The following statement preloads nodes (index blocks) of indexes for the tables **t1** and **t2**:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+-----+-----+-----+-----+
| Table | Op          | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+-----+-----+-----+-----+
```

This statement preloads all index blocks from **t1**. It preloads only blocks for the non-leaf nodes from **t2**.

The syntax of **LOAD INDEX INTO CACHE** enables you to specify that only particular indexes from a table should be preloaded. The current implementation preloads all the table's indexes into the cache, so there is no reason to specify anything other than the table name.

LOAD INDEX INTO CACHE ... IGNORE LEAVES fails unless all indexes in a table have the same block size. (Prior to MySQL 5.1.19, it fails even without **IGNORE LEAVES**.) You can determine index block sizes for a table by using **myisamchk -dv** and checking the **Blocksize** column.

12.5.5.5. RESET Syntax

```
RESET reset_option [, reset_option] ...
```

The **RESET** statement is used to clear the state of various server operations. You must have the **RELOAD** privilege to execute **RESET**.

RESET acts as a stronger version of the **FLUSH** statement. See [Section 12.5.5.2, “FLUSH Syntax”](#).

reset_option can be any of the following:

- **MASTER**

Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file. (Known as **FLUSH MASTER** in versions of MySQL before 3.23.26.) See [Section 12.6.1, “SQL Statements for Controlling Master Servers”](#).

- **QUERY CACHE**

Removes all query results from the query cache.

- **SLAVE**

Makes the slave forget its replication position in the master binary logs. Also resets the relay log by deleting any existing relay log files and beginning a new one. (Known as **FLUSH SLAVE** in versions of MySQL before 3.23.26.) See [Section 12.6.2, “SQL Statements for Controlling Slave Servers”](#).

12.6. Replication Statements

This section describes SQL statements related to replication. One group of statements is used for controlling master servers. The other is used for controlling slave servers.

12.6.1. SQL Statements for Controlling Master Servers

Replication can be controlled through the SQL interface. This section discusses statements for managing master replication servers. [Section 12.6.2, “SQL Statements for Controlling Slave Servers”](#), discusses statements for managing slave servers.

12.6.1.1. PURGE MASTER LOGS Syntax

```
PURGE {MASTER | BINARY} LOGS TO 'log_name'
PURGE {MASTER | BINARY} LOGS BEFORE 'date'
```

Deletes all the binary logs listed in the log index prior to the specified log or date. The logs also are removed from the list recorded in the log index file, so that the given log becomes the first.

Example:

```
PURGE MASTER LOGS TO 'mysql-bin.010';
PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
```

The **BEFORE** variant's *date* argument can be in 'YYYY-MM-DD hh:mm:ss' format. **MASTER** and **BINARY** are synonyms.

This statement is safe to run while slaves are replicating. You do not need to stop them. If you have an active slave that currently is reading one of the logs you are trying to delete, this statement does nothing and fails with an error. However, if a slave is dormant and you happen to purge one of the logs it has yet to read, the slave will be unable to replicate after it comes up.

To safely purge logs, follow this procedure:

1. On each slave server, use **SHOW SLAVE STATUS** to check which log it is reading.
2. Obtain a listing of the binary logs on the master server with **SHOW BINARY LOGS**.
3. Determine the earliest log among all the slaves. This is the target log. If all the slaves are up to date, this is the last log on the list.
4. Make a backup of all the logs you are about to delete. (This step is optional, but always advisable.)
5. Purge all logs up to but not including the target log.

You can also set the `expire_logs_days` system variable to expire binary log files automatically after a given number of days (see [Section 5.2.3, “System Variables”](#)). If you are using replication, you should set the variable no lower than the maximum number of days your slaves might lag behind the master.

12.6.1.2. RESET MASTER Syntax

```
RESET MASTER
```

Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

12.6.1.3. SET SQL_LOG_BIN Syntax

```
SET SQL_LOG_BIN = {0|1}
```

Disables or enables binary logging for the current connection (`SQL_LOG_BIN` is a session variable) if the client has the **SUPER** privilege. The statement is refused with an error if the client does not have that privilege.

12.6.1.4. SHOW BINLOG EVENTS Syntax

```
SHOW BINLOG EVENTS
[IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Shows the events in the binary log. If you do not specify 'log_name', the first binary log is displayed.

The **LIMIT** clause has the same syntax as for the **SELECT** statement. See [Section 12.2.7, “SELECT Syntax”](#).

Note

Issuing a `SHOW BINLOG EVENTS` with no `LIMIT` clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the binary log (which includes all statements executed by the server that modify data). As an alternative to `SHOW BINLOG EVENTS`, use the `mysqlbinlog` utility to save the binary log to a text file for later examination and analysis. See [Section 7.11, “mysqlbinlog — Utility for Processing Binary Log Files”](#).

Note

Events relating to the setting of variables is not included in the output generated when calling `SHOW BINLOG EVENTS`. To get complete coverage of events within a binary log, use `mysqlbinlog`.

12.6.1.5. SHOW BINARY LOGS Syntax

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Lists the binary log files on the server. This statement is used as part of the procedure described in [Section 12.6.1.1, “PURGE MASTER LOGS Syntax”](#), that shows how to determine which logs can be purged.

```
mysql> SHOW BINARY LOGS;
+-----+-----+
| Log_name | File_size |
+-----+-----+
| binlog.000015 | 724935 |
| binlog.000016 | 733481 |
+-----+-----+
```

`SHOW MASTER LOGS` is equivalent to `SHOW BINARY LOGS`.

12.6.1.6. SHOW MASTER STATUS Syntax

```
SHOW MASTER STATUS
```

Provides status information about the binary log files of the master. Example:

```
mysql > SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.003 | 73 | test | manual,mysql |
+-----+-----+-----+-----+
```

12.6.1.7. SHOW SLAVE HOSTS Syntax

```
SHOW SLAVE HOSTS
```

Displays a list of replication slaves currently registered with the master. Only slaves started with the `--report-host=slave_name` option are visible in this list.

The list is displayed on any server (not just the master server). The output looks like this:

```
mysql> SHOW SLAVE HOSTS;
+-----+-----+-----+-----+
| Server_id | Host | Port | Master_id |
+-----+-----+-----+-----+
| 192168010 | iconnect2 | 3306 | 192168011 |
| 1921680101 | athena | 3306 | 192168011 |
+-----+-----+-----+-----+
```

- **Server_id**: The unique server ID of the slave server, as configured in the server's option file, or on the command line with `--server-id=value`.
- **Host**: The host name of the slave server, as configured in the server's option file, or on the command line with `--report-host=value`. Note that this can differ from the machine name as configured in the operating system.

- **Port**: The port the slave server is listening on.
- **Master_id**: The unique server ID of the master server that the slave server is replicating from.

Some MySQL versions report another variable, `Rpl_recovery_rank`. This variable was never used, and was eventually removed.

12.6.2. SQL Statements for Controlling Slave Servers

Replication can be controlled through the SQL interface. This section discusses statements for managing slave replication servers. [Section 12.6.1, “SQL Statements for Controlling Master Servers”](#), discusses statements for managing master servers.

12.6.2.1. CHANGE MASTER TO Syntax

```
CHANGE MASTER TO master_def [, master_def] ...
```

```
master_def:
MASTER_HOST = 'host_name'
MASTER_USER = 'user_name'
MASTER_PASSWORD = 'password'
MASTER_PORT = port_num
MASTER_CONNECT_RETRY = count
MASTER_LOG_FILE = 'master_log_name'
MASTER_LOG_POS = master_log_pos
RELAY_LOG_FILE = 'relay_log_name'
RELAY_LOG_POS = relay_log_pos
MASTER_SSL = {0|1}
MASTER_SSL_CA = 'ca_file_name'
MASTER_SSL_CAPATH = 'ca_directory_name'
MASTER_SSL_CERT = 'cert_file_name'
MASTER_SSL_KEY = 'key_file_name'
MASTER_SSL_CIPHER = 'cipher_list'
MASTER_SSL_VERIFY_SERVER_CERT = {0|1}
```

CHANGE MASTER TO changes the parameters that the slave server uses for connecting to and communicating with the master server. It also updates the contents of the `master.info` and `relay-log.info` files.

`MASTER_USER`, `MASTER_PASSWORD`, `MASTER_SSL`, `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_KEY`, `MASTER_SSL_CIPHER`, and `MASTER_SSL_VERIFY_SERVER_CERT` provide information to the slave about how to connect to its master. `MASTER_SSL_VERIFY_SERVER_CERT` was added in MySQL 5.1.18. It is used as described for the `-ssl-verify-server-cert` option in [Section 5.8.7.3, “SSL Command Options”](#).

The SSL options (`MASTER_SSL`, `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_KEY`, `MASTER_SSL_CIPHER`), and `MASTER_SSL_VERIFY_SERVER_CERT` can be changed even on slaves that are compiled without SSL support. They are saved to the `master.info` file, but are ignored unless you use a server that has SSL support enabled.

If you don't specify a given parameter, it keeps its old value, except as indicated in the following discussion. For example, if the password to connect to your MySQL master has changed, you just need to issue these statements to tell the slave about the new password:

```
STOP SLAVE; -- if replication was running
CHANGE MASTER TO MASTER_PASSWORD='new3cret';
START SLAVE; -- if you want to restart replication
```

There is no need to specify the parameters that do not change (host, port, user, and so forth).

`MASTER_HOST` and `MASTER_PORT` are the hostname (or IP address) of the master host and its TCP/IP port. Note that if `MASTER_HOST` is equal to `localhost`, then, like in other parts of MySQL, the port number might be ignored (if Unix socket files can be used, for example).

If you specify `MASTER_HOST` or `MASTER_PORT`, the slave assumes that the master server is different from before (even if you specify a host or port value that is the same as the current value.) In this case, the old values for the master binary log name and position are considered no longer applicable, so if you do not specify `MASTER_LOG_FILE` and `MASTER_LOG_POS` in the statement, `MASTER_LOG_FILE=''` and `MASTER_LOG_POS=4` are silently appended to it.

`MASTER_LOG_FILE` and `MASTER_LOG_POS` are the coordinates at which the slave I/O thread should begin reading from the master the next time the thread starts. If you specify either of them, you cannot specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. If neither of `MASTER_LOG_FILE` or `MASTER_LOG_POS` are specified, the slave uses the last coordinates of the *slave SQL thread* before **CHANGE MASTER** was issued. This ensures that there is no discontinuity in replication, even if the slave SQL thread was late compared to the slave I/O thread, when you merely want to change, say, the password to use.

CHANGE MASTER *deletes all relay log files* and starts a new one, unless you specify **RELAY_LOG_FILE** or **RELAY_LOG_POS**. In that case, relay logs are kept; the **relay_log_purge** global variable is set silently to 0.

CHANGE MASTER is useful for setting up a slave when you have the snapshot of the master and have recorded the log and the offset corresponding to it. After loading the snapshot into the slave, you can run **CHANGE MASTER TO MASTER_LOG_FILE='log_name_on_master', MASTER_LOG_POS=log_offset_on_master** on the slave.

The following example changes the master and master's binary log coordinates. This is used when you want to set up the slave to replicate the master:

```
CHANGE MASTER TO
  MASTER_HOST='master2.mycompany.com',
  MASTER_USER='replication',
  MASTER_PASSWORD='bigs3cret',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='master2-bin.001',
  MASTER_LOG_POS=4,
  MASTER_CONNECT_RETRY=10;
```

The next example shows an operation that is less frequently employed. It is used when the slave has relay logs that you want it to execute again for some reason. To do this, the master need not be reachable. You need only use **CHANGE MASTER TO** and start the SQL thread (**START SLAVE SQL_THREAD**):

```
CHANGE MASTER TO
  RELAY_LOG_FILE='slave-relay-bin.006',
  RELAY_LOG_POS=4025;
```

You can even use the second operation in a non-replication setup with a standalone, non-slave server for recovery following a crash. Suppose that your server has crashed and you have restored a backup. You want to replay the server's own binary logs (not relay logs, but regular binary logs), named (for example) **myhost-bin.***. First, make a backup copy of these binary logs in some safe place, in case you don't exactly follow the procedure below and accidentally have the server purge the binary logs. Use **SET GLOBAL relay_log_purge=0** for additional safety. Then start the server without the **--log-bin** option. Instead, use the **--replicate-same-server-id**, **--relay-log=myhost-bin** (to make the server believe that these regular binary logs are relay logs) and **--skip-slave-start** options. After the server starts, issue these statements:

```
CHANGE MASTER TO
  RELAY_LOG_FILE='myhost-bin.153',
  RELAY_LOG_POS=410,
  MASTER_HOST='some_dummy_string';
START SLAVE SQL_THREAD;
```

The server reads and executes its own binary logs, thus achieving crash recovery. Once the recovery is finished, run **STOP SLAVE**, shut down the server, delete the **master.info** and **relay-log.info** files, and restart the server with its original options.

Specifying the **MASTER_HOST** option (even with a dummy value) is required to make the server think it is a slave.

12.6.2.2. LOAD DATA FROM MASTER Syntax

```
LOAD DATA FROM MASTER
```

This feature is deprecated. We recommend not using it anymore. It is subject to removal in a future version of MySQL.

Since the current implementation of **LOAD DATA FROM MASTER** and **LOAD TABLE FROM MASTER** is very limited, these statements are deprecated in versions 4.1 of MySQL and above. We will introduce a more advanced technique (called “online backup”) in a future version. That technique will have the additional advantage of working with more storage engines.

For MySQL 5.1 and earlier, the recommended alternative solution to using **LOAD DATA FROM MASTER** or **LOAD TABLE FROM MASTER** is using **mysqldump** or **mysqlhotcopy**. The latter requires Perl and two Perl modules (**DBI** and **DBD:mysql**) and works for **MyISAM** and **ARCHIVE** tables only. With **mysqldump**, you can create SQL dumps on the master and pipe (or copy) these to a **mysql** client on the slave. This has the advantage of working for all storage engines, but can be quite slow, since it works using **SELECT**.

This statement takes a snapshot of the master and copies it to the slave. It updates the values of **MASTER_LOG_FILE** and **MASTER_LOG_POS** so that the slave starts replicating from the correct position. Any table and database exclusion rules specified with the **--replicate-*-do-*** and **--replicate-*-ignore-*** options are honored. **--replicate-rewrite-db** is *not* taken into account because a user could use this option to set up a non-unique mapping such as **--replicate-rewrite-db="db1->db3"** and **--replicate-rewrite-db="db2->db3"**, which would confuse the slave when loading tables from the master.

Use of this statement is subject to the following conditions:

- It works only for **MyISAM** tables. Attempting to load a non-**MyISAM** table results in the following error:

```
ERROR 1189 (08S01): Net error reading from master
```

- It acquires a global read lock on the master while taking the snapshot, which prevents updates on the master during the load operation.

If you are loading large tables, you might have to increase the values of `net_read_timeout` and `net_write_timeout` on both the master and slave servers. See [Section 5.2.3, “System Variables”](#).

Note that **LOAD DATA FROM MASTER** does *not* copy any tables from the **mysql** database. This makes it easy to have different users and privileges on the master and the slave.

To use **LOAD DATA FROM MASTER**, the replication account that is used to connect to the master must have the **RELOAD** and **SUPER** privileges on the master and the **SELECT** privilege for all master tables you want to load. All master tables for which the user does not have the **SELECT** privilege are ignored by **LOAD DATA FROM MASTER**. This is because the master hides them from the user: **LOAD DATA FROM MASTER** calls **SHOW DATABASES** to know the master databases to load, but **SHOW DATABASES** returns only databases for which the user has some privilege. See [Section 12.5.4.12, “SHOW DATABASES Syntax”](#). On the slave side, the user that issues **LOAD DATA FROM MASTER** must have privileges for dropping and creating the databases and tables that are copied.

12.6.2.3. **LOAD TABLE *tbl_name* FROM MASTER** Syntax

```
LOAD TABLE tbl_name FROM MASTER
```

This feature is deprecated. We recommend not using it anymore. It is subject to removal in a future version of MySQL.

Since the current implementation of **LOAD DATA FROM MASTER** and **LOAD TABLE FROM MASTER** is very limited, these statements are deprecated in versions 4.1 of MySQL and above. We will introduce a more advanced technique (called “online backup”) in a future version. That technique will have the additional advantage of working with more storage engines.

For MySQL 5.1 and earlier, the recommended alternative solution to using **LOAD DATA FROM MASTER** or **LOAD TABLE FROM MASTER** is using **mysqldump** or **mysqlhotcopy**. The latter requires Perl and two Perl modules (**DBI** and **DBD:mysql**) and works for **MyISAM** and **ARCHIVE** tables only. With **mysqldump**, you can create SQL dumps on the master and pipe (or copy) these to a **mysql** client on the slave. This has the advantage of working for all storage engines, but can be quite slow, since it works using **SELECT**.

Transfers a copy of the table from the master to the slave. This statement is implemented mainly debugging **LOAD DATA FROM MASTER** operations. To use **LOAD TABLE**, the account used for connecting to the master server must have the **RELOAD** and **SUPER** privileges on the master and the **SELECT** privilege for the master table to load. On the slave side, the user that issues **LOAD TABLE FROM MASTER** must have privileges for dropping and creating the table.

The conditions for **LOAD DATA FROM MASTER** apply here as well. For example, **LOAD TABLE FROM MASTER** works only for **MyISAM** tables. The timeout notes for **LOAD DATA FROM MASTER** apply as well.

12.6.2.4. **MASTER_POS_WAIT()** Syntax

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos)
```

This is actually a function, not a statement. It is used to ensure that the slave has read and executed events up to a given position in the master's binary log. See [Section 11.11.4, “Miscellaneous Functions”](#), for a full description.

12.6.2.5. **RESET SLAVE** Syntax

```
RESET SLAVE
```

RESET SLAVE makes the slave forget its replication position in the master's binary logs. This statement is meant to be used for a clean start: It deletes the `master.info` and `relay-log.info` files, all the relay logs, and starts a new relay log.

Note

All relay logs are deleted, even if they have not been completely executed by the slave SQL thread. (This is a condition likely to exist on a replication slave if you have issued a **STOP SLAVE** statement or if the slave is highly loaded.)

Connection information stored in the `master.info` file is immediately reset using any values specified in the corresponding startup options. This information includes values such as master host, master port, master user, and master password. If the slave SQL thread was in the middle of replicating temporary tables when it was stopped, and **RESET SLAVE** is issued, these replicated temporary tables are deleted on the slave.

12.6.2.6. SET GLOBAL SQL_SLAVE_SKIP_COUNTER Syntax

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = N
```

This statement skips the next *N* events from the master. This is useful for recovering from replication stops caused by a statement.

This statement is valid only when the slave thread is not running. Otherwise, it produces an error.

12.6.2.7. SHOW SLAVE STATUS Syntax

```
SHOW SLAVE STATUS
```

This statement provides status information on essential parameters of the slave threads. If you issue this statement using the `mysql` client, you can use a `\G` statement terminator rather than a semicolon to obtain a more readable vertical layout:

```
mysql> SHOW SLAVE STATUS\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
        Master_Host: localhost
        Master_User: root
        Master_Port: 3306
        Connect_Retry: 3
        Master_Log_File: gbichot-bin.005
        Read_Master_Log_Pos: 79
        Relay_Log_File: gbichot-relay-bin.005
        Relay_Log_Pos: 548
        Relay_Master_Log_File: gbichot-bin.005
        Slave_IO_Running: Yes
        Slave_SQL_Running: Yes
        Replicate_Do_DB:
        Replicate_Ignore_DB:
          Last_Errno: 0
          Last_Error:
          Skip_Counter: 0
        Exec_Master_Log_Pos: 79
        Relay_Log_Space: 552
        Until_Condition: None
        Until_Log_File:
        Until_Log_Pos: 0
        Master_SSL_Allowed: No
        Master_SSL_CA_File:
        Master_SSL_CA_Path:
        Master_SSL_Cert:
        Master_SSL_Cipher:
        Master_SSL_Key:
        Seconds_Behind_Master: 8
Master_SSL_Verify_Server_Cert: No
          Last_IO_Errno: 0
          Last_IO_Error:
          Last_SQL_Errno: 0
          Last_SQL_Error:
```

SHOW SLAVE STATUS returns the following fields:

- **Slave_IO_State**

A copy of the **State** field of the output of **SHOW PROCESSLIST** for the slave I/O thread. This tells you what the thread is doing: trying to connect to the master, waiting for events from the master, reconnecting to the master, and so on. Possible states are listed in [Section 15.4.1, “Replication Implementation Details”](#). It is necessary to check this field for older versions of MySQL which allowed the thread to continue running while unsuccessfully trying to connect to the master. If it is running, there is no problem; if it is not, you can find the error in the **Last_Error** field (described below).

- **Master_Host**

The current master host.

- **Master_User**

The current user used to connect to the master.

- **Master_Port**

The current master port.

- **Connect_Retry**

The current value of the `--master-connect-retry` option.

- **Master_Log_File**

The name of the master binary log file from which the I/O thread is currently reading.

- **Read_Master_Log_Pos**

The position up to which the I/O thread has read in the current master binary log.

- **Relay_Log_File**

The name of the relay log file from which the SQL thread is currently reading and executing.

- **Relay_Log_Pos**

The position up to which the SQL thread has read and executed in the current relay log.

- **Relay_Master_Log_File**

The name of the master binary log file containing the most recent event executed by the SQL thread.

- **Slave_IO_Running**

Whether the I/O thread is started and has connected successfully to the master. For older versions of MySQL (prior to 4.1.14 and 5.0.12) **Slave_IO_Running** is **YES** if the I/O thread is started, even if the slave hasn't connected to the master yet.

- **Slave_SQL_Running**

Whether the SQL thread is started.

- **Replicate_Do_DB, Replicate_Ignore_DB**

The lists of databases that were specified with the `--replicate-do-db` and `--replicate-ignore-db` options, if any.

- **Replicate_Do_Table, Replicate_Ignore_Table, Replicate_Wild_Do_Table, Replicate_Wild_Ignore_Table**

The lists of tables that were specified with the `--replicate-do-table`, `--replicate-ignore-table`, `--replicate-wild-do-table`, and `--replicate-wild-ignore-table` options, if any.

- **Last_Errno, Last_Error**

As of MySQL 5.1.20, these columns are aliases for **Last_SQLErrno** and **Last_SQLError**. Before 5.1.20, they indicate the error number and error message returned by the most recently executed query. An error number of 0 and message of the empty string mean “no error.” If the **Last_Error** value is not empty, it also appears as a message in the slave's error log.

- **Skip_Counter**

The most recently used value for **SQL_SLAVE_SKIP_COUNTER**.

- **Exec_Master_Log_Pos**

The position of the last event executed by the SQL thread from the master's binary log (**Relay_Master_Log_File**). (Re-

`lay_Master_Log_File`, `Exec_Master_Log_Pos`) in the master's binary log corresponds to (`Relay_Log_File`, `Relay_Log_Pos`) in the relay log.

- `Relay_Log_Space`

The total combined size of all existing relay logs.

- `Until_Condition`, `Until_Log_File`, `Until_Log_Pos`

The values specified in the `UNTIL` clause of the `START SLAVE` statement.

`Until_Condition` has these values:

- `None` if no `UNTIL` clause was specified
- `Master` if the slave is reading until a given position in the master's binary logs
- `Relay` if the slave is reading until a given position in its relay logs

`Until_Log_File` and `Until_Log_Pos` indicate the log filename and position values that define the point at which the SQL thread stops executing.

- `Master_SSL_Allowed`, `Master_SSL_CA_File`, `Master_SSL_CA_Path`, `Master_SSL_Cert`, `Master_SSL_Cipher`, `Master_SSL_Key`

These fields show the SSL parameters used by the slave to connect to the master, if any.

`Master_SSL_Allowed` has these values:

- `Yes` if an SSL connection to the master is permitted
- `No` if an SSL connection to the master is not permitted
- `Ignored` if an SSL connection is permitted but the slave server does not have SSL support enabled

The values of the other SSL-related fields correspond to the values of the `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_CIPHER`, `MASTER_SSL_KEY`, and `MASTER_SSL_VERIFY_SERVER_CERT` options to the `CHANGE MASTER` statement. See [Section 12.6.2.1, “CHANGE MASTER TO Syntax”](#). `MASTER_SSL_VERIFY_SERVER_CERT` was added in MySQL 5.1.18.

- `Seconds_Behind_Master`

This field is an indication of how “late” the slave is:

- When the slave SQL thread is actively running (processing updates), this field is the number of seconds that have elapsed since the timestamp of the most recent event on the master executed by that thread.
- When the SQL thread has caught up to the slave I/O thread and goes idle waiting for more events from the I/O thread, this field is zero.

In essence, this field measures the time difference in seconds between the slave SQL thread and the slave I/O thread.

If the network connection between master and slave is fast, the slave I/O thread is very close to the master, so this field is a good approximation of how late the slave SQL thread is compared to the master. If the network is slow, this is *not* a good approximation; the slave SQL thread may quite often be caught up with the slow-reading slave I/O thread, so `Seconds_Behind_Master` often shows a value of 0, even if the I/O thread is late compared to the master. In other words, *this column is useful only for fast networks*.

This time difference computation works even though the master and slave do not have identical clocks (the clock difference is computed when the slave I/O thread starts, and assumed to remain constant from then on). `Seconds_Behind_Master` is `NULL` (which means “unknown”) if the slave SQL thread is not running, or if the slave I/O thread is not running or not connected to master. For example if the slave I/O thread is sleeping for the number of seconds given by the `--master-connect-retry` option before reconnecting, `NULL` is shown, as the slave cannot know what the master is doing, and so cannot say reliably how late it is.

This field has one limitation. The timestamp is preserved through replication, which means that, if a master M1 is itself a slave of M0, any event from M1's binlog which originates in replicating an event from M0's binlog has the timestamp of that event. This enables MySQL to replicate `TIMESTAMP` successfully. However, the drawback for `Seconds_Behind_Master` is that if M1 also

receives direct updates from clients, the value randomly deviates, because sometimes the last M1's event is from M0 and sometimes it is the most recent timestamp from a direct update.

- **Last_IO_Errno, Last_IO_Error**

The error number and error message of the last error that caused the I/O thread to stop. An error number of 0 and message of the empty string mean “no error.” If the **Last_IO_Error** value is not empty, it also appears as a message in the slave's error log. These columns were added in MySQL 5.1.20.

- **Last_SQL_Errno, Last_SQL_Error**

The error number and error message of the last error that caused the SQL thread to stop. An error number of 0 and message of the empty string mean “no error.” If the **Last_IO_Error** value is not empty, it also appears as a message in the slave's error log. These columns were added in MySQL 5.1.20.

Example:

```
Last_SQL_Errno: 1051
Last_SQL_Error: error 'Unknown table 'z'' on query 'drop table z'
```

The message indicates that the table **z** existed on the master and was dropped there, but it did not exist on the slave, so **DROP TABLE** failed on the slave. (This might occur, for example, if you forget to copy the table to the slave when setting up replication.)

12.6.2.8. START SLAVE Syntax

```
START SLAVE [thread_type [, thread_type] ... ]
START SLAVE [SQL_THREAD] UNTIL
  MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
START SLAVE [SQL_THREAD] UNTIL
  RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos

thread_type: IO_THREAD | SQL_THREAD
```

START SLAVE with no *thread_type* options starts both of the slave threads. The I/O thread reads queries from the master server and stores them in the relay log. The SQL thread reads the relay log and executes the queries. **START SLAVE** requires the **SUPER** privilege.

If **START SLAVE** succeeds in starting the slave threads, it returns without any error. However, even in that case, it might be that the slave threads start and then later stop (for example, because they do not manage to connect to the master or read its binary logs, or some other problem). **START SLAVE** does not warn you about this. You must check the slave's error log for error messages generated by the slave threads, or check that they are running satisfactorily with **SHOW SLAVE STATUS**.

You can add **IO_THREAD** and **SQL_THREAD** options to the statement to name which of the threads to start.

An **UNTIL** clause may be added to specify that the slave should start and run until the SQL thread reaches a given point in the master binary logs or in the slave relay logs. When the SQL thread reaches that point, it stops. If the **SQL_THREAD** option is specified in the statement, it starts only the SQL thread. Otherwise, it starts both slave threads. If the SQL thread is running, the **UNTIL** clause is ignored and a warning is issued.

For an **UNTIL** clause, you must specify both a log filename and position. Do not mix master and relay log options.

Any **UNTIL** condition is reset by a subsequent **STOP SLAVE** statement, a **START SLAVE** statement that includes no **UNTIL** clause, or a server restart.

The **UNTIL** clause can be useful for debugging replication, or to cause replication to proceed until just before the point where you want to avoid having the slave replicate a statement. For example, if an unwise **DROP TABLE** statement was executed on the master, you can use **UNTIL** to tell the slave to execute up to that point but no farther. To find what the event is, use **mysqlbinlog** with the master logs or slave relay logs, or by using a **SHOW BINLOG EVENTS** statement.

If you are using **UNTIL** to have the slave process replicated queries in sections, it is recommended that you start the slave with the **-skip-slave-start** option to prevent the SQL thread from running when the slave server starts. It is probably best to use this option in an option file rather than on the command line, so that an unexpected server restart does not cause it to be forgotten.

The **SHOW SLAVE STATUS** statement includes output fields that display the current values of the **UNTIL** condition.

In old versions of MySQL (before 4.0.5), this statement was called **SLAVE START**. This usage is still accepted in MySQL 5.1 for backward compatibility, but is deprecated.

12.6.2.9. STOP SLAVE Syntax

```
STOP SLAVE [thread_type [, thread_type] ... ]  
thread_type: IO_THREAD | SQL_THREAD
```

Stops the slave threads. **STOP SLAVE** requires the **SUPER** privilege.

Like **START SLAVE**, this statement may be used with the **IO_THREAD** and **SQL_THREAD** options to name the thread or threads to be stopped.

In old versions of MySQL (before 4.0.5), this statement was called **SLAVE STOP**. This usage is still accepted in MySQL 5.1 for backward compatibility, but is deprecated.

12.7. SQL Syntax for Prepared Statements

MySQL 5.1 provides support for server-side prepared statements. This support takes advantage of the efficient client/server binary protocol implemented in MySQL 4.1, provided that you use an appropriate client programming interface. Candidate interfaces include the MySQL C API client library (for C programs), MySQL Connector/J (for Java programs), and MySQL Connector/NET. For example, the C API provides a set of function calls that make up its prepared statement API. See [Section 25.2.4, “C API Prepared Statements”](#). Other language interfaces can provide support for prepared statements that use the binary protocol by linking in the C client library, one example being the [mysql extension](#), available in PHP 5.0 and later.

An alternative SQL interface to prepared statements is available. This interface is not as efficient as using the binary protocol through a prepared statement API, but requires no programming because it is available directly at the SQL level:

- You can use it when no programming interface is available to you.
- You can use it from any program that allows you to send SQL statements to the server to be executed, such as the [mysql](#) client program.
- You can use it even if the client is using an old version of the client library. The only requirement is that you be able to connect to a server that is recent enough to support SQL syntax for prepared statements.

SQL syntax for prepared statements is intended to be used for situations such as these:

- You want to test how prepared statements work in your application before coding it.
- An application has problems executing prepared statements and you want to determine interactively what the problem is.
- You want to create a test case that describes a problem you are having with prepared statements, so that you can file a bug report.
- You need to use prepared statements but do not have access to a programming API that supports them.

SQL syntax for prepared statements is based on three SQL statements:

- **PREPARE** *stmt_name* **FROM** *preparable_stmt*

The **PREPARE** statement prepares a statement and assigns it a name, *stmt_name*, by which to refer to the statement later. Statement names are not case sensitive. *preparable_stmt* is either a string literal or a user variable that contains the text of the statement. The text must represent a single SQL statement, not multiple statements. Within the statement, ‘?’ characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The ‘?’ characters should not be enclosed within quotes, even if you intend to bind them to string values. Parameter markers can be used only where data values should appear, not for SQL keywords, identifiers, and so forth.

If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name

exists.

The scope of a prepared statement is the client session within which it is created. Other clients cannot see it.

- `EXECUTE stmt_name [USING @var_name [, @var_name] ...]`

After preparing a statement, you execute it with an `EXECUTE` statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a `USING` clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the `USING` clause must name exactly as many variables as the number of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing different variables to it or setting the variables to different values before each execution.

- `{DEALLOCATE | DROP} PREPARE stmt_name`

To deallocate a prepared statement, use the `DEALLOCATE PREPARE` statement. Attempting to execute a prepared statement after deallocating it results in an error.

If you terminate a client session without deallocating a previously prepared statement, the server deallocates it automatically.

The following SQL statements can be used in prepared statements: `CREATE TABLE`, `DELETE`, `DO`, `INSERT`, `REPLACE`, `SELECT`, `SET`, `UPDATE`, and most `SHOW` statements.

As of MySQL 5.1.10, the following additional statements are supported:

```
ANALYZE TABLE
OPTIMIZE TABLE
REPAIR TABLE
```

As of MySQL 5.1.12, the following additional statements are supported:

```
CACHE INDEX
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
{CREATE | RENAME | DROP} DATABASE
{CREATE | RENAME | DROP} USER
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
      | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES}
GRANT
REVOKE
KILL
LOAD INDEX INTO CACHE
RESET {MASTER | SLAVE | QUERY CACHE}
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {AUTHORS | CONTRIBUTORS | WARNINGS | ERRORS}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
INSTALL PLUGIN
UNINSTALL PLUGIN
```

Other statements are not yet supported.

Statements not allowed in SQL prepared statements are generally also not permitted in stored routines. Any exceptions to this rule are noted in [Chapter 19, Stored Procedures and Functions](#).

The following examples show two equivalent ways of preparing a statement that computes the hypotenuse of a triangle given the lengths of the two sides.

The first example shows how to create a prepared statement by using a string literal to supply the text of the statement:

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|          5 |
```

```
+-----+
mysql> DEALLOCATE PREPARE stmt1;
```

The second example is similar, but supplies the text of the statement as a user variable:

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|          10 |
+-----+
mysql> DEALLOCATE PREPARE stmt2;
```

Placeholders can be used for the arguments of the [LIMIT](#) clause when using prepared statements. See [Section 12.2.7, “SELECT Syntax”](#).

SQL syntax for prepared statements cannot be used in nested fashion. That is, a statement passed to [PREPARE](#) cannot itself be a [PREPARE](#), [EXECUTE](#), or [DEALLOCATE PREPARE](#) statement.

SQL syntax for prepared statements is distinct from using prepared statement API calls. For example, you cannot use the [mysql_stmt_prepare\(\)](#) C API function to prepare a [PREPARE](#), [EXECUTE](#), or [DEALLOCATE PREPARE](#) statement.

SQL syntax for prepared statements can be used within stored procedures, but not in stored functions or triggers. However, a cursor cannot be used for a dynamic statement that is prepared and executed with [PREPARE](#) and [EXECUTE](#). The statement for a cursor is checked at cursor creation time, so the statement cannot be dynamic.

SQL syntax for prepared statements does not support multi-statements (that is, multiple statements within a single string separated by `;` characters).

Before MySQL 5.1.17, prepared statements do not use the query cache. As of 5.1.17, prepared statements use the query cache under the conditions described in [Section 6.5.4.1, “How the Query Cache Operates”](#).

Chapter 13. Storage Engines

MySQL supports several storage engines that act as handlers for different table types. MySQL storage engines include both those that handle transaction-safe tables and those that handle non-transaction-safe tables:

With MySQL 5.1, MySQL AB has introduced a new pluggable storage engine architecture that allows storage engines to be loaded into and unloaded from a running MySQL server.

This chapter describes each of the MySQL storage engines except for **NDB Cluster**, which is covered in [Chapter 16, MySQL Cluster](#). It also contains a description of the pluggable storage engine architecture (see [Section 13.1, “Overview of MySQL Storage Engine Architecture”](#)).

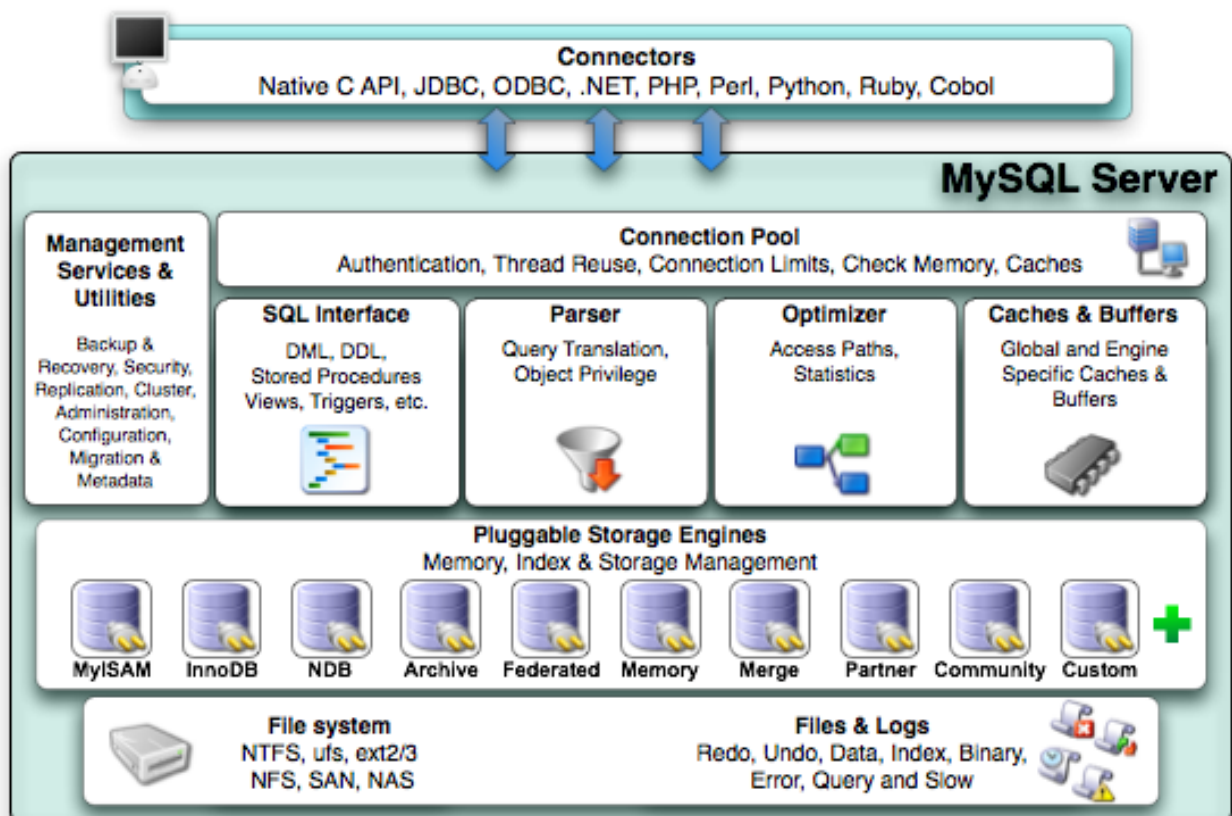
For answers to some commonly asked questions about MySQL storage engines, see [Section A.2, “MySQL 5.1 FAQ — Storage Engines”](#).

13.1. Overview of MySQL Storage Engine Architecture

The MySQL pluggable storage engine architecture allows a database professional to select a specialized storage engine for a particular application need while being completely shielded from the need to manage any specific application coding requirements. The MySQL server architecture isolates the application programmer and DBA from all of the low-level implementation details at the storage level, providing a consistent and easy application model and API. Thus, although there are different capabilities across different storage engines, the application is shielded from these differences.

The MySQL pluggable storage engine architecture is shown in [Figure 13.1, “The MySQL architecture using pluggable storage engines”](#).

Figure 13.1. The MySQL architecture using pluggable storage engines



The pluggable storage engine architecture provides a standard set of management and support services that are common among all underlying storage engines. The storage engines themselves are the components of the database server that actually perform actions on the underlying data that is maintained at the physical server level.

This efficient and modular architecture provides huge benefits for those wishing to specifically target a particular application need — such as data warehousing, transaction processing, or high availability situations — while enjoying the advantage of utilizing a set of interfaces and services that are independent of any one storage engine.

The application programmer and DBA interact with the MySQL database through Connector APIs and service layers that are above the storage engines. If application changes bring about requirements that demand the underlying storage engine change, or that one or more additional storage engines be added to support new needs, no significant coding or process changes are required to make things work. The MySQL server architecture shields the application from the underlying complexity of the storage engine by presenting a consistent and easy-to-use API that applies across storage engines.

13.1.1. The Common Database Server Layer

A MySQL pluggable storage engine is the component in the MySQL database server that is responsible for performing the actual data I/O operations for a database as well as enabling and enforcing certain feature sets that target a specific application need. A major benefit of using specific storage engines is that you are only delivered the features needed for a particular application, and therefore you have less system overhead in the database, with the end result being more efficient and higher database performance. This is one of the reasons that MySQL has always been known to have such high performance, matching or beating proprietary monolithic databases in industry standard benchmarks.

From a technical perspective, what are some of the unique supporting infrastructure components that are in a storage engine? Some of the key feature differentiations include:

- *Concurrency* — some applications have more granular lock requirements (such as row-level locks) than others. Choosing the right locking strategy can reduce overhead and therefore improve overall performance. This area also includes support for capabilities such as multi-version concurrency control or “snapshot” read.
- *Transaction Support* — Not every application needs transactions, but for those that do, there are very well defined requirements such as ACID compliance and more.
- *Referential Integrity* — The need to have the server enforce relational database referential integrity through DDL defined foreign keys.
- *Physical Storage* — This involves everything from the overall page size for tables and indexes as well as the format used for storing data to physical disk.
- *Index Support* — Different application scenarios tend to benefit from different index strategies. Each storage engine generally has its own indexing methods, although some (such as B-tree indexes) are common to nearly all engines.
- *Memory Caches* — Different applications respond better to some memory caching strategies than others, so although some memory caches are common to all storage engines (such as those used for user connections or MySQL's high-speed Query Cache), others are uniquely defined only when a particular storage engine is put in play.
- *Performance Aids* — This includes multiple I/O threads for parallel operations, thread concurrency, database checkpointing, bulk insert handling, and more.
- *Miscellaneous Target Features* — This may include support for geospatial operations, security restrictions for certain data manipulation operations, and other similar features.

Each set of the pluggable storage engine infrastructure components are designed to offer a selective set of benefits for a particular application. Conversely, avoiding a set of component features helps reduce unnecessary overhead. It stands to reason that understanding a particular application's set of requirements and selecting the proper MySQL storage engine can have a dramatic impact on overall system efficiency and performance.

13.1.2. Pluggable Storage Engine Architecture

With MySQL 5.1, MySQL AB has introduced a new pluggable storage engine architecture that allows storage engines to be loaded into and unloaded from a running MySQL server.

Plugging in a Storage Engine

Before a storage engine can be used, the storage engine plugin shared library must be loaded into MySQL using the `INSTALL PLUGIN` statement. For example, if the `EXAMPLE` engine plugin is named `ha_example` and the shared library is named `ha_example.so`, you load it with the following statement:

```
mysql> INSTALL PLUGIN ha_example SONAME 'ha_example.so';
```

To install a pluggable storage engine, the plugin file must be located in the MySQL plugin directory, and the user issuing the `INSTALL PLUGIN` statement must have `INSERT` privileges for the `mysql.plugin` table.

The shared library must be located in the MySQL server plugin directory, the location of which is given by the `plugin_dir` system variable.

Unplugging a Storage Engine

To unplug a storage engine, use the `UNINSTALL PLUGIN` statement:

```
mysql> UNINSTALL PLUGIN ha_example;
```

If you unplug a storage engine that is needed by existing tables, those tables become inaccessible, but will still be present on disk (where applicable). Ensure that there are no tables using a storage engine before you unplug the storage engine.

13.2. Supported Storage Engines

MySQL 5.1 supports the following storage engines:

- **MyISAM** — The default MySQL storage engine and the one that is used the most in Web, data warehousing, and other application environments. **MyISAM** is supported in all MySQL configurations, and is the default storage engine unless you have configured MySQL to use a different one by default.
- **InnoDB** — Used for transaction processing applications, and sports a number of features including ACID transaction support and foreign keys. **InnoDB** is included by default in all MySQL 5.1 binary distributions. In source distributions, you can enable or disable either engine by configuring MySQL as you like.
- **Memory** — Stores all data in RAM for extremely fast access in environments that require quick lookups of reference and other like data. This engine was formerly known as the **HEAP** engine.
- **Merge** — Allows a MySQL DBA or developer to logically group a series of identical **MyISAM** tables and reference them as one object. Good for VLDB environments such as data warehousing.
- **Archive** — Provides the perfect solution for storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.
- **Federated** — Offers the ability to link separate MySQL servers to create one logical database from many physical servers. Very good for distributed or data mart environments.
- **NDB** — The Clustered database engine that is particularly suited for applications with high performance lookup needs that also require the highest possible degree of uptime and availability.
- **CSV** — The CSV storage engine stores data in text files using comma-separated values format. You can use the CSV engine to easily exchange data between other software and applications that can import and export in CSV format.
- **Blackhole** — The Blackhole storage engine accepts but does not store data and retrievals always return an empty set. The functionality can be used in distributed database design where data is automatically replicated, but not stored locally.
- **Example** — The Example storage engine is “stub” engine that does nothing. You can create tables with this engine, but no data can be stored in them or retrieved from them. The purpose of this engine is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

This chapter describes each of the MySQL storage engines except for **NDB Cluster**, which is covered in [Chapter 16, MySQL Cluster](#).

It is important to remember that you are not restricted to using the same storage engine for an entire server or schema: you can use a different storage engine for each table in your schema.

13.2.1. Choosing a Storage Engine

The various storage engines provided with MySQL are designed with different use-cases in mind. To use the pluggable storage architecture effectively, it is good to have an idea of the benefits and drawbacks of the various storage engines. The following table provides an overview of some storage engines provided with MySQL:

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Storage limits	256TB	Yes	64TB	No	384EB ^a
Transactions	No	No	Yes	No	Yes
Locking granularity	Table	Table	Row	Row	Row
MVCC (snapshot read)	No	No	Yes	Yes	No
Geospatial datatype support	Yes	No	Yes	Yes	Yes
Geospatial indexing support	Yes	No	No	No	No
B-tree indexes	Yes	Yes	Yes	No	Yes
Hash indexes	No	Yes	No	No	Yes
Full-text search indexes	Yes	No	No	No	No
Clustered indexes	No	No	Yes	No	No
Data caches	No	N/A	Yes	No	Yes
Index caches	Yes	N/A	Yes	No	Yes
Compressed data	Yes ^b	No	No	Yes	No
Encrypted data ^c	Yes	Yes	Yes	Yes	Yes
Cluster database support	No	No	No	No	Yes
Replication support ^d	Yes	Yes	Yes	Yes	Yes
Foreign key support	No	No	Yes	No	No
Backup / point-in-time recovery ^e	Yes	Yes	Yes	Yes	Yes
Query cache support	Yes	Yes	Yes	Yes	Yes
Update statistics for data dictionary	Yes	Yes	Yes	Yes	Yes

^a EB = exabyte (1024 * 1024 terabyte)

^b Compressed MyISAM tables are only supported when using the compressed row format. Tables using the compressed row format with MyISAM are read-only.

^c Implemented in the server (via encryption functions), rather than in the storage engine.

^d Implemented in the server, rather than in the storage engine

^e Implemented in the server, rather than in the storage engine

13.2.2. Comparing Transaction and Non-Transaction Engines

Transaction-safe tables (TSTs) have several advantages over non-transaction-safe tables (NTSTs):

- They are safer. Even if MySQL crashes or you get hardware problems, you can get your data back, either by automatic recovery or from a backup plus the transaction log.
- You can combine many statements and accept them all at the same time with the **COMMIT** statement (if autocommit is disabled).
- You can execute **ROLLBACK** to ignore your changes (if autocommit is disabled).
- If an update fails, all of your changes are reverted. (With non-transaction-safe tables, all changes that have taken place are permanent.)
- Transaction-safe storage engines can provide better concurrency for tables that get many updates concurrently with reads.

You can combine transaction-safe and non-transaction-safe tables in the same statements to get the best of both worlds. However, although MySQL supports several transaction-safe storage engines, for best results, you should not mix different storage engines within a transaction with autocommit disabled. For example, if you do this, changes to non-transaction-safe tables still are committed immedi-

ately and cannot be rolled back. For information about this and other problems that can occur in transactions that use mixed storage engines, see [Section 12.4.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#).

Non-transaction-safe tables have several advantages of their own, all of which occur because there is no transaction overhead:

- Much faster
- Lower disk space requirements
- Less memory required to perform updates

13.2.3. Other Storage Engines

Other storage engines may be available from third parties and community members that have used the Custom Storage Engine interface.

You can find more information on the list of third party storage engines on the [MySQL Forge Storage Engines](#) page.

Note

Third party engines are not supported by MySQL. For further information, documentation, installation guides, bug reporting or for any help or assistance with these engines, please contact the developer of the engine directly.

Third party engines that are known to be available include the following; please see the MySQL Forge links provided for more information:

- **PrimeBase XT (PBXT)** — PBXT has been designed for modern, web-based, high concurrency environments.
- **RitmarkFS** — RitmarkFS allows you to access and manipulate the filesystem using SQL queries. RitmarkFS also supports filesystem replication and directory change tracking.
- **Distributed Data Engine** — The Distributed Data Engine is an Open Source project that is dedicated to provide a Storage Engine for distributed data according to workload statistics.
- **mdbtools** — A pluggable storage engine that allows read-only access to Microsoft Access `.mdb` database files.
- **solidDB for MySQL** — solidDB Storage Engine for MySQL is an open source, transactional storage engine for MySQL Server. It is designed for mission-critical implementations that require a robust, transactional database. solidDB Storage Engine for MySQL is a multi-threaded storage engine that supports full ACID compliance with all expected transaction isolation levels, row-level locking, and Multi-Version Concurrency Control (MVCC) with non-blocking reads and writes.

For more information on developing a customer storage engine that can be used with the Pluggable Storage Engine Architecture, see [Writing a Custom Storage Engine](#) on [MySQL Forge](#).

13.3. Setting the Storage Engine

When you create a new table, you can specify which storage engine to use by adding an **ENGINE** table option to the **CREATE TABLE** statement:

```
CREATE TABLE t (i INT) ENGINE = INNODB;
```

If you omit the **ENGINE** or **TYPE** option, the default storage engine is used. Normally, this is **MyISAM**, but you can change it by using the `--default-storage-engine` or `--default-table-type` server startup option, or by setting the `default-storage-engine` or `default-table-type` option in the `my.cnf` configuration file.

You can set the default storage engine to be used during the current session by setting the `storage_engine` variable:

```
SET storage_engine=MYISAM;
```

When MySQL is installed on Windows using the MySQL Configuration Wizard, the **InnoDB** storage engine can be selected as the default instead of **MyISAM**. See [Section 2.3.4.6, “The Database Usage Dialog”](#).

To convert a table from one storage engine to another, use an `ALTER TABLE` statement that indicates the new engine:

```
ALTER TABLE t ENGINE = MYISAM;
```

See [Section 12.1.10, “CREATE TABLE Syntax”](#), and [Section 12.1.4, “ALTER TABLE Syntax”](#).

If you try to use a storage engine that is not compiled in or that is compiled in but deactivated, MySQL instead creates a table using the default storage engine, usually `MyISAM`. This behavior is convenient when you want to copy tables between MySQL servers that support different storage engines. (For example, in a replication setup, perhaps your master server supports transactional storage engines for increased safety, but the slave servers use only non-transactional storage engines for greater speed.)

This automatic substitution of the default storage engine for unavailable engines can be confusing for new MySQL users. A warning is generated whenever a storage engine is automatically changed.

For new tables, MySQL always creates an `.frm` file to hold the table and column definitions. The table's index and data may be stored in one or more other files, depending on the storage engine. The server creates the `.frm` file above the storage engine level. Individual storage engines create any additional files required for the tables that they manage. If a table name contains special characters, the names for the table files contain encoded versions of those characters as described in [Section 8.2.3, “Mapping of Identifiers to File-names”](#).

A database may contain tables of different types. That is, tables need not all be created with the same storage engine.

13.4. The `MyISAM` Storage Engine

`MyISAM` is the default storage engine. It is based on the older `ISAM` code but has many useful extensions. (Note that MySQL 5.1 does *not* support `ISAM`.)

Each `MyISAM` table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table format. The data file has an `.MYD` (`MYData`) extension. The index file has an `.MYI` (`MYIndex`) extension.

To specify explicitly that you want a `MyISAM` table, indicate that with an `ENGINE` table option:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

Normally, it is unnecessary to use `ENGINE` to specify the `MyISAM` storage engine. `MyISAM` is the default engine unless the default has been changed. To ensure that `MyISAM` is used in situations where the default might have been changed, include the `ENGINE` option explicitly.

You can check or repair `MyISAM` tables with the `mysqlcheck` client or `myisamchk` utility. You can also compress `MyISAM` tables with `myisampack` to take up much less space. See [Section 7.12, “mysqlcheck — A Table Maintenance and Repair Program”](#), [Section 5.9.4.1, “Using myisamchk for Crash Recovery”](#), and [Section 7.7, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#).

`MyISAM` tables have the following characteristics:

- All data values are stored with the low byte first. This makes the data machine and operating system independent. The only requirements for binary portability are that the machine uses two's-complement signed integers and IEEE floating-point format. These requirements are widely used among mainstream machines. Binary compatibility might not be applicable to embedded systems, which sometimes have peculiar processors.

There is no significant speed penalty for storing data low byte first; the bytes in a table row normally are unaligned and it takes little more processing to read an unaligned byte in order than in reverse order. Also, the code in the server that fetches column values is not time critical compared to other code.

- All numeric key values are stored with the high byte first to allow better index compression.
- Large files (up to 63-bit file length) are supported on filesystems and operating systems that support large files.
- There is a limit of 2^{32} (~4.295E+09) rows in a `MyISAM` table. If you build MySQL with the `--with-big-tables` option, the row limitation is increased to $(2^{32})^2$ (1.844E+19) rows. See [Section 2.9.2, “Typical configure Options”](#). Binary distributions for Unix and Linux are built with this option.

- The maximum number of indexes per **MyISAM** table is 64. This can be changed by recompiling. Beginning with MySQL 5.1.4, you can configure the build by invoking **configure** with the **--with-max-indexes=N** option, where *N* is the maximum number of indexes to permit per **MyISAM** table. *N* must be less than or equal to 128. Before MySQL 5.1.4, you must change the source.

The maximum number of columns per index is 16.

- The maximum key length is 1000 bytes. This can also be changed by changing the source and recompiling. For the case of a key longer than 250 bytes, a larger key block size than the default of 1024 bytes is used.
- When rows are inserted in sorted order (as when you are using an **AUTO_INCREMENT** column), the index tree is split so that the high node only contains one key. This improves space utilization in the index tree.
- Internal handling of one **AUTO_INCREMENT** column per table is supported. **MyISAM** automatically updates this column for **INSERT** and **UPDATE** operations. This makes **AUTO_INCREMENT** columns faster (at least 10%). Values at the top of the sequence are not reused after being deleted. (When an **AUTO_INCREMENT** column is defined as the last column of a multiple-column index, reuse of values deleted from the top of a sequence does occur.) The **AUTO_INCREMENT** value can be reset with **ALTER TABLE** or **myisamchk**.
- Dynamic-sized rows are much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.
- **MyISAM** supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can **INSERT** new rows into it at the same time that other threads are reading from the table. A free block can occur as a result of deleting rows or an update of a dynamic length row with more data than its current contents. When all free blocks are used up (filled in), future inserts become concurrent again. See [Section 6.3.3, “Concurrent Inserts”](#).
- You can put the data file and index file on different directories to get more speed with the **DATA DIRECTORY** and **INDEX DIRECTORY** table options to **CREATE TABLE**. See [Section 12.1.10, “CREATE TABLE Syntax”](#).
- **BLOB** and **TEXT** columns can be indexed.
- **NULL** values are allowed in indexed columns. This takes 0–1 bytes per key.
- Each character column can have a different character set. See [Chapter 9, Character Set Support](#).
- There is a flag in the **MyISAM** index file that indicates whether the table was closed correctly. If **mysqld** is started with the **-myisam-recover** option, **MyISAM** tables are automatically checked when opened, and are repaired if the table wasn't closed properly.
- **myisamchk** marks tables as checked if you run it with the **--update-state** option. **myisamchk --fast** checks only those tables that don't have this mark.
- **myisamchk --analyze** stores statistics for portions of keys, as well as for entire keys.
- **myisampack** can pack **BLOB** and **VARCHAR** columns.

MyISAM also supports the following features:

- Support for a true **VARCHAR** type; a **VARCHAR** column starts with a length stored in one or two bytes.
- Tables with **VARCHAR** columns may have fixed or dynamic row length.
- The sum of the lengths of the **VARCHAR** and **CHAR** columns in a table may be up to 64KB.
- Arbitrary length **UNIQUE** constraints.

Additional resources

- A forum dedicated to the **MyISAM** storage engine is available at <http://forums.mysql.com/list.php?21>.

13.4.1. **MyISAM** Startup Options

The following options to `mysqld` can be used to change the behavior of `MyISAM` tables. For additional information, see [Section 5.2.2, “Command Options”](#).

- `--myisam-recover=mode`

Set the mode for automatic recovery of crashed `MyISAM` tables.

- `--delay-key-write=ALL`

Don't flush key buffers between writes for any `MyISAM` table.

Note

If you do this, you should not access `MyISAM` tables from another program (such as from another MySQL server or with `myisamchk`) when the tables are in use. Doing so risks index corruption. Using `--external-locking` does not eliminate this risk.

The following system variables affect the behavior of `MyISAM` tables. For additional information, see [Section 5.2.3, “System Variables”](#).

- `bulk_insert_buffer_size`

The size of the tree cache used in bulk insert optimization. **Note:** This is a limit *per thread*!

- `myisam_max_sort_file_size`

The maximum size of the temporary file that MySQL is allowed to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

- `myisam_sort_buffer_size`

Set the size of the buffer used when recovering tables.

Automatic recovery is activated if you start `mysqld` with the `--myisam-recover` option. In this case, when the server opens a `MyISAM` table, it checks whether the table is marked as crashed or whether the open count variable for the table is not 0 and you are running the server with external locking disabled. If either of these conditions is true, the following happens:

- The server checks the table for errors.
- If the server finds an error, it tries to do a fast table repair (with sorting and without re-creating the data file).
- If the repair fails because of an error in the data file (for example, a duplicate-key error), the server tries again, this time re-creating the data file.
- If the repair still fails, the server tries once more with the old repair option method (write row by row without sorting). This method should be able to repair any type of error and has low disk space requirements.

MySQL Enterprise

Subscribers to MySQL Enterprise Monitor receive notification if the `--myisam-recover` option has not been set. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

If the recovery wouldn't be able to recover all rows from previously completed statements and you didn't specify `FORCE` in the value of the `--myisam-recover` option, automatic repair aborts with an error message in the error log:

```
Error: Couldn't repair table: test.g00pages
```

If you specify `FORCE`, a warning like this is written instead:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

Note that if the automatic recovery value includes [BACKUP](#), the recovery process creates files with names of the form `tbl_name-datetime.BAK`. You should have a [cron](#) script that automatically moves these files from the database directories to backup media.

13.4.2. Space Needed for Keys

[MyISAM](#) tables use B-tree indexes. You can roughly calculate the size for the index file as $(key\_length+4)/0.67$, summed over all keys. This is for the worst case when all keys are inserted in sorted order and the table doesn't have any compressed keys.

String indexes are space compressed. If the first index part is a string, it is also prefix compressed. Space compression makes the index file smaller than the worst-case figure if a string column has a lot of trailing space or is a [VARCHAR](#) column that is not always used to the full length. Prefix compression is used on keys that start with a string. Prefix compression helps if there are many strings with an identical prefix.

In [MyISAM](#) tables, you can also prefix compress numbers by specifying the [PACK_KEYS=1](#) table option when you create the table. Numbers are stored with the high byte first, so this helps when you have many integer keys that have an identical prefix.

13.4.3. [MyISAM](#) Table Storage Formats

[MyISAM](#) supports three different storage formats. Two of them, fixed and dynamic format, are chosen automatically depending on the type of columns you are using. The third, compressed format, can be created only with the [myisampack](#) utility.

When you use [CREATE TABLE](#) or [ALTER TABLE](#) for a table that has no [BLOB](#) or [TEXT](#) columns, you can force the table format to [FIXED](#) or [DYNAMIC](#) with the [ROW_FORMAT](#) table option.

You can decompress tables by specifying [ROW_FORMAT=DEFAULT](#) with [ALTER TABLE](#).

See [Section 12.1.10, “CREATE TABLE Syntax”](#), for information about [ROW_FORMAT](#).

13.4.3.1. Static (Fixed-Length) Table Characteristics

Static format is the default for [MyISAM](#) tables. It is used when the table contains no variable-length columns ([VARCHAR](#), [VARBINARY](#), [BLOB](#), or [TEXT](#)). Each row is stored using a fixed number of bytes.

Of the three [MyISAM](#) storage formats, static format is the simplest and most secure (least subject to corruption). It is also the fastest of the on-disk formats due to the ease with which rows in the data file can be found on disk: To look up a row based on a row number in the index, multiply the row number by the row length to calculate the row position. Also, when scanning a table, it is very easy to read a constant number of rows with each disk read operation.

The security is evidenced if your computer crashes while the MySQL server is writing to a fixed-format [MyISAM](#) file. In this case, [my-isamchk](#) can easily determine where each row starts and ends, so it can usually reclaim all rows except the partially written one. Note that [MyISAM](#) table indexes can always be reconstructed based on the data rows.

Note

Fixed-length row format is only available for tables without [BLOB](#) or [TEXT](#) columns. Creating a table with these columns with an explicit [ROW_FORMAT](#) clause will not raise an error or warning; the format specification will be ignored.

Static-format tables have these characteristics:

- [CHAR](#) and [VARCHAR](#) columns are space-padded to the specified column width, although the column type is not altered. [BINARY](#) and [VARBINARY](#) columns are padded with `0x00` bytes to the column width.
- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because rows are located in fixed positions.
- Reorganization is unnecessary unless you delete a huge number of rows and want to return free disk space to the operating system. To do this, use [OPTIMIZE TABLE](#) or [myisamchk -r](#).
- Usually require more disk space than dynamic-format tables.

13.4.3.2. Dynamic Table Characteristics

Dynamic storage format is used if a [MyISAM](#) table contains any variable-length columns ([VARCHAR](#), [VARBINARY](#), [BLOB](#), or [TEXT](#)), or if the table was created with the [ROW_FORMAT=DYNAMIC](#) table option.

Dynamic format is a little more complex than static format because each row has a header that indicates how long it is. A row can become fragmented (stored in non-contiguous pieces) when it is made longer as a result of an update.

You can use [OPTIMIZE TABLE](#) or [myisamchk -r](#) to defragment a table. If you have fixed-length columns that you access or change frequently in a table that also contains some variable-length columns, it might be a good idea to move the variable-length columns to other tables just to avoid fragmentation.

Dynamic-format tables have these characteristics:

- All string columns are dynamic except those with a length less than four.
- Each row is preceded by a bitmap that indicates which columns contain the empty string (for string columns) or zero (for numeric columns). Note that this does not include columns that contain [NULL](#) values. If a string column has a length of zero after trailing space removal, or a numeric column has a value of zero, it is marked in the bitmap and not saved to disk. Non-empty strings are saved as a length byte plus the string contents.
- Much less disk space usually is required than for fixed-length tables.
- Each row uses only as much space as is required. However, if a row becomes larger, it is split into as many pieces as are required, resulting in row fragmentation. For example, if you update a row with information that extends the row length, the row becomes fragmented. In this case, you may have to run [OPTIMIZE TABLE](#) or [myisamchk -r](#) from time to time to improve performance. Use [myisamchk -ei](#) to obtain table statistics.
- More difficult than static-format tables to reconstruct after a crash, because rows may be fragmented into many pieces and links (fragments) may be missing.
- The expected row length for dynamic-sized rows is calculated using the following expression:

```
3
+ (number of columns + 7) / 8
+ (number of char columns)
+ (packed size of numeric columns)
+ (length of strings)
+ (number of NULL columns + 7) / 8
```

There is a penalty of 6 bytes for each link. A dynamic row is linked whenever an update causes an enlargement of the row. Each new link is at least 20 bytes, so the next enlargement probably goes in the same link. If not, another link is created. You can find the number of links using [myisamchk -ed](#). All links may be removed with [OPTIMIZE TABLE](#) or [myisamchk -r](#).

13.4.3.3. Compressed Table Characteristics

Compressed storage format is a read-only format that is generated with the [mysampack](#) tool. Compressed tables can be uncompressed with [myisamchk](#).

Compressed tables have the following characteristics:

- Compressed tables take very little disk space. This minimizes disk usage, which is helpful when using slow disks (such as CD-ROMs).
- Each row is compressed separately, so there is very little access overhead. The header for a row takes up one to three bytes depending on the biggest row in the table. Each column is compressed differently. There is usually a different Huffman tree for each column. Some of the compression types are:
 - Suffix space compression.
 - Prefix space compression.
 - Numbers with a value of zero are stored using one bit.

- If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a **BIGINT** column (eight bytes) can be stored as a **TINYINT** column (one byte) if all its values are in the range from **-128** to **127**.
- If a column has only a small set of possible values, the data type is converted to **ENUM**.
- A column may use any combination of the preceding compression types.
- Can be used for fixed-length or dynamic-length rows.

Note. While a compressed table is read-only, and you cannot therefore update or add rows in the table, DDL (Data Definition Language) operations are still valid. For example, you may still use **DROP** to drop the table, and **TRUNCATE** to empty the table.

13.4.4. MyISAM Table Problems

The file format that MySQL uses to store data has been extensively tested, but there are always circumstances that may cause database tables to become corrupted. The following discussion describes how this can happen and how to handle it.

13.4.4.1. Corrupted MyISAM Tables

Even though the **MyISAM** table format is very reliable (all changes to a table made by an SQL statement are written before the statement returns), you can still get corrupted tables if any of the following events occur:

- The **mysqld** process is killed in the middle of a write.
- An unexpected computer shutdown occurs (for example, the computer is turned off).
- Hardware failures.
- You are using an external program (such as **myisamchk**) to modify a table that is being modified by the server at the same time.
- A software bug in the MySQL or **MyISAM** code.

Typical symptoms of a corrupt table are:

- You get the following error while selecting data from the table:

```
Incorrect key file for table: '...'. Try to repair it
```

- Queries don't find rows in the table or return incomplete results.

You can check the health of a **MyISAM** table using the **CHECK TABLE** statement, and repair a corrupted **MyISAM** table with **REPAIR TABLE**. When **mysqld** is not running, you can also check or repair a table with the **myisamchk** command. See [Section 12.5.2.3, “CHECK TABLE Syntax”](#), [Section 12.5.2.6, “REPAIR TABLE Syntax”](#), and [Section 7.5, “myisamchk — MyISAM Table-Maintenance Utility”](#).

If your tables become corrupted frequently, you should try to determine why this is happening. The most important thing to know is whether the table became corrupted as a result of a server crash. You can verify this easily by looking for a recent **restarted mysqld** message in the error log. If there is such a message, it is likely that table corruption is a result of the server dying. Otherwise, corruption may have occurred during normal operation. This is a bug. You should try to create a reproducible test case that demonstrates the problem. See [Section B.1.4.2, “What to Do If MySQL Keeps Crashing”](#), and [MySQL Internals: Porting](#).

MySQL Enterprise

Find out about problems before they occur. Subscribe to the MySQL Enterprise Monitor for expert advice about the state of your servers. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

13.4.4.2. Problems from Tables Not Being Closed Properly

Each **MyISAM** index file (**.MYI** file) has a counter in the header that can be used to check whether a table has been closed properly. If you get the following warning from **CHECK TABLE** or **myisamchk**, it means that this counter has gone out of sync:

```
clients are using or haven't closed the table properly
```

This warning doesn't necessarily mean that the table is corrupted, but you should at least check the table.

The counter works as follows:

- The first time a table is updated in MySQL, a counter in the header of the index files is incremented.
- The counter is not changed during further updates.
- When the last instance of a table is closed (because a [FLUSH TABLES](#) operation was performed or because there is no room in the table cache), the counter is decremented if the table has been updated at any point.
- When you repair the table or check the table and it is found to be okay, the counter is reset to zero.
- To avoid problems with interaction with other processes that might check the table, the counter is not decremented on close if it was zero.

In other words, the counter can become incorrect only under these conditions:

- A [MyISAM](#) table is copied without first issuing [LOCK TABLES](#) and [FLUSH TABLES](#).
- MySQL has crashed between an update and the final close. (Note that the table may still be okay, because MySQL always issues writes for everything between each statement.)
- A table was modified by [myisamchk --recover](#) or [myisamchk --update-state](#) at the same time that it was in use by [mysqld](#).
- Multiple [mysqld](#) servers are using the table and one server performed a [REPAIR TABLE](#) or [CHECK TABLE](#) on the table while it was in use by another server. In this setup, it is safe to use [CHECK TABLE](#), although you might get the warning from other servers. However, [REPAIR TABLE](#) should be avoided because when one server replaces the data file with a new one, this is not known to the other servers.

In general, it is a bad idea to share a data directory among multiple servers. See [Section 5.12, “Running Multiple MySQL Servers on the Same Machine”](#), for additional discussion.

13.5. The [InnoDB](#) Storage Engine

13.5.1. [InnoDB](#) Overview

[InnoDB](#) provides MySQL with a transaction-safe ([ACID](#) compliant) storage engine that has commit, rollback, and crash recovery capabilities. [InnoDB](#) does locking on the row level and also provides an Oracle-style consistent non-locking read in [SELECT](#) statements. These features increase multi-user concurrency and performance. There is no need for lock escalation in [InnoDB](#) because row-level locks fit in very little space. [InnoDB](#) also supports [FOREIGN KEY](#) constraints. You can freely mix [InnoDB](#) tables with tables from other MySQL storage engines, even within the same statement.

[InnoDB](#) has been designed for maximum performance when processing large data volumes. Its CPU efficiency is probably not matched by any other disk-based relational database engine.

Fully integrated with MySQL Server, the [InnoDB](#) storage engine maintains its own buffer pool for caching data and indexes in main memory. [InnoDB](#) stores its tables and indexes in a tablespace, which may consist of several files (or raw disk partitions). This is different from, for example, [MyISAM](#) tables where each table is stored using separate files. [InnoDB](#) tables can be of any size even on operating systems where file size is limited to 2GB.

[InnoDB](#) is included in binary distributions by default. The Windows Essentials installer makes [InnoDB](#) the MySQL default storage engine on Windows.

[InnoDB](#) is used in production at numerous large database sites requiring high performance. The famous Internet news site Slashdot.org runs on [InnoDB](#). Mytrix, Inc. stores over 1TB of data in [InnoDB](#), and another site handles an average load of 800 inserts/updates per second in [InnoDB](#).

InnoDB is published under the same GNU GPL License Version 2 (of June 1991) as MySQL. For more information on MySQL licensing, see <http://www.mysql.com/company/legal/licensing/>.

Additional resources

- A forum dedicated to the **InnoDB** storage engine is available at <http://forums.mysql.com/list.php?22>.

13.5.2. InnoDB Contact Information

Contact information for Innobase Oy, producer of the **InnoDB** engine:

```
Web site: http://www.innodb.com/
Email: <sales@innodb.com>
Phone: +358-9-6969 3250 (office)
      +358-40-5617367 (mobile)

Innbase Oy Inc.
World Trade Center Helsinki
Aleksanterinkatu 17
P.O.Box 800
00101 Helsinki
Finland
```

13.5.3. InnoDB Configuration

The **InnoDB** storage engine is enabled by default. If you don't want to use **InnoDB** tables, you can add the `skip-innodb` option to your MySQL option file.

Note

InnoDB provides MySQL with a transaction-safe (**ACID** compliant) storage engine that has commit, rollback, and crash recovery capabilities. **However, it cannot do so** if the underlying operating system or hardware does not work as advertised. Many operating systems or disk subsystems may delay or reorder write operations to improve performance. On some operating systems, the very system call that should wait until all unwritten data for a file has been flushed — `fsync()` — might actually return before the data has been flushed to stable storage. Because of this, an operating system crash or a power outage may destroy recently committed data, or in the worst case, even corrupt the database because of write operations having been reordered. If data integrity is important to you, you should perform some “pull-the-plug” tests before using anything in production. On Mac OS X 10.3 and up, **InnoDB** uses a special `fcntl()` file flush method. Under Linux, it is advisable to **disable the write-back cache**.

On ATAPI hard disks, a command such `hdparm -W0 /dev/hda` may work to disable the write-back cache. **Beware that some drives or disk controllers may be unable to disable the write-back cache.**

Two important disk-based resources managed by the **InnoDB** storage engine are its tablespace data files and its log files.

Note

If you specify no **InnoDB** configuration options, MySQL creates an auto-extending 10MB data file named `ibdata1` and two 5MB log files named `ib_logfile0` and `ib_logfile1` in the MySQL data directory. To get good performance, you should explicitly provide **InnoDB** parameters as discussed in the following examples. Naturally, you should edit the settings to suit your hardware and requirements.

Note

It is not a good idea to configure **InnoDB** to use datafiles or logfiles on NFS volumes. Otherwise, the files might be locked by other processes and become unavailable for use by MySQL.

MySQL Enterprise

For advice on settings suitable to your specific circumstances, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

The examples shown here are representative. See [Section 13.5.4, “InnoDB Startup Options and System Variables”](#) for additional information about **InnoDB**-related configuration parameters.

To set up the **InnoDB** tablespace files, use the `innodb_data_file_path` option in the `[mysqld]` section of the `my.cnf` option file. On Windows, you can use `my.ini` instead. The value of `innodb_data_file_path` should be a list of one or more data file specifications. If you name more than one data file, separate them by semicolon (‘;’) characters:

```
innodb_data_file_path=datafile_spec1[;datafile_spec2]...
```

For example, a setting that explicitly creates a tablespace having the same characteristics as the default is as follows:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend
```

This setting configures a single 10MB data file named `ibdata1` that is auto-extending. No location for the file is given, so by default, **InnoDB** creates it in the MySQL data directory.

Sizes are specified using **M** or **G** suffix letters to indicate units of MB or GB.

A tablespace containing a fixed-size 50MB data file named `ibdata1` and a 50MB auto-extending file named `ibdata2` in the data directory can be configured like this:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

The full syntax for a data file specification includes the filename, its size, and several optional attributes:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

The `autoextend` attribute and those following can be used only for the last data file in the `innodb_data_file_path` line.

If you specify the `autoextend` option for the last data file, **InnoDB** extends the data file if it runs out of free space in the tablespace. The increment is 8MB at a time by default. It can be modified by changing the `innodb_autoextend_increment` system variable.

If the disk becomes full, you might want to add another data file on another disk. Instructions for reconfiguring an existing tablespace are given in [Section 13.5.7, “Adding and Removing InnoDB Data and Log Files”](#).

InnoDB is not aware of the filesystem maximum file size, so be cautious on filesystems where the maximum file size is a small value such as 2GB. To specify a maximum size for an auto-extending data file, use the `max` attribute. The following configuration allows `ibdata1` to grow up to a limit of 500MB:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend:max:500M
```

InnoDB creates tablespace files in the MySQL data directory by default. To specify a location explicitly, use the `innodb_data_home_dir` option. For example, to use two files named `ibdata1` and `ibdata2` but create them in the `/ibdata` directory, configure **InnoDB** like this:

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

Note

InnoDB does not create directories, so make sure that the `/ibdata` directory exists before you start the server. This is also true of any log file directories that you configure. Use the Unix or DOS `mkdir` command to create any necessary directories.

InnoDB forms the directory path for each data file by textually concatenating the value of `innodb_data_home_dir` to the data file name, adding a pathname separator (slash or backslash) between values if necessary. If the `innodb_data_home_dir` option is not mentioned in `my.cnf` at all, the default value is the “dot” directory `./`, which means the MySQL data directory. (The MySQL server changes its current working directory to its data directory when it begins executing.)

If you specify `innodb_data_home_dir` as an empty string, you can specify absolute paths for the data files listed in the `innodb_data_file_path` value. The following example is equivalent to the preceding one:

```
[mysqld]
innodb_data_home_dir =
```

```
innodb_data_file_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoextend
```

A simple `my.cnf` example. Suppose that you have a computer with 128MB RAM and one hard disk. The following example shows possible configuration parameters in `my.cnf` or `my.ini` for InnoDB, including the `autoextend` attribute. The example suits most users, both on Unix and Windows, who do not want to distribute InnoDB data files and log files onto several disks. It creates an auto-extending data file `ibdata1` and two InnoDB log files `ib_logfile0` and `ib_logfile1` in the MySQL data directory.

```
[mysqld]
# You can write your other MySQL server options here
# ...
# Data files must be able to hold your data and indexes.
# Make sure that you have enough free disk space.
innodb_data_file_path = ibdata1:10M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory
innodb_buffer_pool_size=70M
innodb_additional_mem_pool_size=10M
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=20M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
```

Make sure that the MySQL server has the proper access rights to create files in the data directory. More generally, the server must have access rights in any directory where it needs to create data files or log files.

Note that data files must be less than 2GB in some filesystems. The combined size of the log files must be less than 4GB. The combined size of data files must be at least 10MB.

When you create an InnoDB tablespace for the first time, it is best that you start the MySQL server from the command prompt. InnoDB then prints the information about the database creation to the screen, so you can see what is happening. For example, on Windows, if `mysqld` is located in `C:\Program Files\MySQL\MySQL Server 5.1\bin`, you can start it like this:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld" --console
```

If you do not send server output to the screen, check the server's error log to see what InnoDB prints during the startup process.

See [Section 13.5.5, “Creating the InnoDB Tablespace”](#), for an example of what the information displayed by InnoDB should look like.

You can place InnoDB options in the `[mysqld]` group of any option file that your server reads when it starts. The locations for option files are described in [Section 4.3.2, “Using Option Files”](#).

If you installed MySQL on Windows using the installation and configuration wizards, the option file will be the `my.ini` file located in your MySQL installation directory. See [Section 2.3.4.14, “The Location of the my.ini File”](#).

If your PC uses a boot loader where the `C:` drive is not the boot drive, your only option is to use the `my.ini` file in your Windows directory (typically `C:\WINDOWS`). You can use the `SET` command at the command prompt in a console window to print the value of `WINDIR`:

```
C:\> SET WINDIR
windir=C:\WINDOWS
```

If you want to make sure that `mysqld` reads options only from a specific file, you can use the `--defaults-file` option as the first option on the command line when starting the server:

```
mysqld --defaults-file=your_path_to_my.cnf
```

An advanced `my.cnf` example. Suppose that you have a Linux computer with 2GB RAM and three 60GB hard disks at directory paths `/`, `/dr2` and `/dr3`. The following example shows possible configuration parameters in `my.cnf` for InnoDB.

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#
# Data files must be able to hold your data and indexes
innodb_data_file_path = /db/ibdata1:2000M;/dr2/db/ibdata2:2000M:autoextend
#
```

```
# Set buffer pool size to 50-80% of your computer's memory,
# but make sure on Linux x86 total memory usage is < 2GB
innodb_buffer_pool_size=1G
innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir = /dr3/iblogs
#
innodb_log_files_in_group = 2
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=250M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
innodb_lock_wait_timeout=50
#
# Uncomment the next lines if you want to use them
#innodb_thread_concurrency=5
```

In some cases, database performance improves if all the data is not placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. The example illustrates how to do this. It places the two data files on different disks and places the log files on the third disk. **InnoDB** fills the tablespace beginning with the first data file. You can also use raw disk partitions (raw devices) as **InnoDB** data files, which may speed up I/O. See [Section 13.5.3.2, “Using Raw Devices for the Shared Tablespace”](#).

Warning

On 32-bit GNU/Linux x86, you must be careful not to set memory usage too high. **glibc** may allow the process heap to grow over thread stacks, which crashes your server. It is a risk if the value of the following expression is close to or exceeds 2GB:

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

Each thread uses a stack (often 2MB, but only 256KB in MySQL AB binaries) and in the worst case also uses `sort_buffer_size` + `read_buffer_size` additional memory.

By compiling MySQL yourself, you can use up to 64GB of physical memory in 32-bit Windows. See the description for `innodb_buffer_pool_awesome_mem_mb` in [Section 13.5.4, “InnoDB Startup Options and System Variables”](#).

How to tune other `mysqld` server parameters? The following values are typical and suit most users:

```
[mysqld]
skip-external-locking
max_connections=200
read_buffer_size=1M
sort_buffer_size=1M
#
# Set key_buffer to 5 - 50% of your RAM depending on how much
# you use MyISAM tables, but keep key_buffer_size + InnoDB
# buffer pool size < 80% of your RAM
key_buffer_size=va lue
```

13.5.3.1. Using Per-Table Tablespaces

You can store each **InnoDB** table and its indexes in its own file. This feature is called “multiple tablespaces” because in effect each table has its own tablespace.

Using multiple tablespaces can be beneficial to users who want to move specific tables to separate physical disks or who wish to restore backups of single tables quickly without interrupting the use of the remaining **InnoDB** tables.

You can enable multiple tablespaces by adding this line to the `[mysqld]` section of `my.cnf`:

```
[mysqld]
innodb_file_per_table
```

After restarting the server, **InnoDB** stores each newly created table into its own file `tbl_name.ibd` in the database directory where the table belongs. This is similar to what the **MyISAM** storage engine does, but **MyISAM** divides the table into a data file `tbl_name.MYD` and the index file `tbl_name.MYI`. For **InnoDB**, the data and the indexes are stored together in the `.ibd` file. The `tbl_name.frm` file is still created as usual.

If you remove the `innodb_file_per_table` line from `my.cnf` and restart the server, **InnoDB** creates tables inside the shared tablespace files again.

`innodb_file_per_table` affects only table creation, not access to existing tables. If you start the server with this option, new tables are created using `.ibd` files, but you can still access tables that exist in the shared tablespace. If you remove the option and restart the server, new tables are created in the shared tablespace, but you can still access any tables that were created using multiple tablespaces.

Note

InnoDB always needs the shared tablespace because it puts its internal data dictionary and undo logs there. The `.ibd` files are not sufficient for **InnoDB** to operate.

Note

You cannot freely move `.ibd` files between database directories as you can with **MyISAM** table files. This is because the table definition that is stored in the **InnoDB** shared tablespace includes the database name, and because **InnoDB** must preserve the consistency of transaction IDs and log sequence numbers.

To move an `.ibd` file and the associated table from one database to another, use a `RENAME TABLE` statement:

```
RENAME TABLE db1.tbl_name TO db2.tbl_name;
```

If you have a “clean” backup of an `.ibd` file, you can restore it to the MySQL installation from which it originated as follows:

1. Issue this `ALTER TABLE` statement:

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

Caution

This statement deletes the current `.ibd` file.

2. Put the backup `.ibd` file back in the proper database directory.
3. Issue this `ALTER TABLE` statement:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

In this context, a “clean” `.ibd` file backup means:

- There are no uncommitted modifications by transactions in the `.ibd` file.
- There are no unmerged insert buffer entries in the `.ibd` file.
- Purge has removed all delete-marked index records from the `.ibd` file.
- `mysqld` has flushed all modified pages of the `.ibd` file from the buffer pool to the file.

You can make a clean backup `.ibd` file using the following method:

1. Stop all activity from the `mysqld` server and commit all transactions.
2. Wait until `SHOW ENGINE INNODB STATUS` shows that there are no active transactions in the database, and the main thread status of **InnoDB** is `Waiting for server activity`. Then you can make a copy of the `.ibd` file.

Another method for making a clean copy of an `.ibd` file is to use the commercial **InnoDB Hot Backup** tool:

1. Use **InnoDB Hot Backup** to back up the **InnoDB** installation.
2. Start a second **mysqld** server on the backup and let it clean up the **.ibd** files in the backup.

13.5.3.2. Using Raw Devices for the Shared Tablespace

You can use raw disk partitions as data files in the shared tablespace. By using a raw disk, you can perform non-buffered I/O on Windows and on some Unix systems without filesystem overhead, which may improve performance.

When you create a new data file, you must put the keyword **newraw** immediately after the data file size in **innodb_data_file_path**. The partition must be at least as large as the size that you specify. Note that 1MB in **InnoDB** is 1024×1024 bytes, whereas 1MB in disk specifications usually means 1,000,000 bytes.

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
```

The next time you start the server, **InnoDB** notices the **newraw** keyword and initializes the new partition. However, do not create or change any **InnoDB** tables yet. Otherwise, when you next restart the server, **InnoDB** reinitializes the partition and your changes are lost. (As a safety measure **InnoDB** prevents users from modifying data when any partition with **newraw** is specified.)

After **InnoDB** has initialized the new partition, stop the server, change **newraw** in the data file specification to **raw**:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:5Graw;/dev/hdd2:2Graw
```

Then restart the server and **InnoDB** allows changes to be made.

On Windows, you can allocate a disk partition as a data file like this:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=\\.\D: :10Gnewraw
```

The **\\.** corresponds to the Windows syntax of **\\.** for accessing physical drives.

When you use raw disk partitions, be sure that they have permissions that allow read and write access by the account used for running the MySQL server.

13.5.4. InnoDB Startup Options and System Variables

This section describes the **InnoDB**-related command options and system variables. System variables that are true or false can be enabled at server startup by naming them, or disabled by using a **skip-** prefix. For example, to enable or disable **InnoDB** checksums, you can use **--innodb_checksums** or **--skip-innodb_checksums** on the command line, or **innodb_checksums** or **skip-innodb_checksums** in an option file. System variables that take a numeric value can be specified as **--var_name=value** on the command line or as **var_name=value** in option files. For more information on specifying options and system variables, see [Section 4.3, “Specifying Program Options”](#). Many of the system variables can be changed at runtime (see [Section 5.2.4.2, “Dynamic System Variables”](#)).

MySQL Enterprise

The MySQL Enterprise Monitor provides expert advice on **InnoDB** start-up options and related system variables. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

InnoDB command options:

- **--innodb**
Enables the **InnoDB** storage engine, if the server was compiled with **InnoDB** support. Use **--skip-innodb** to disable **InnoDB**.
- **--innodb_status_file**
Causes **InnoDB** to create a file named **<datadir>/innodb_status.<pid>** in the MySQL data directory. **InnoDB** periodic-

ally writes the output of `SHOW ENGINE INNODB STATUS` to this file.

InnoDB system variables:

- `innodb_additional_mem_pool_size`

The size in bytes of a memory pool InnoDB uses to store data dictionary information and other internal data structures. The more tables you have in your application, the more memory you need to allocate here. If InnoDB runs out of memory in this pool, it starts to allocate memory from the operating system and writes warning messages to the MySQL error log. The default value is 1MB.

- `innodb_autoextend_increment`

The increment size (in MB) for extending the size of an auto-extending tablespace when it becomes full. The default value is 8.

- `innodb_buffer_pool_ave_mem_mb`

The size of the buffer pool (in MB), if it is placed in the AWE memory. This is relevant only in 32-bit Windows. If your 32-bit Windows operating system supports more than 4GB memory, using so-called “Address Windowing Extensions,” you can allocate the InnoDB buffer pool into the AWE physical memory using this variable. The maximum possible value for this variable is 63000. If it is greater than 0, `innodb_buffer_pool_size` is the window in the 32-bit address space of `mysqld` where InnoDB maps that AWE memory. A good value for `innodb_buffer_pool_size` is 500MB.

To take advantage of AWE memory, you will need to recompile MySQL yourself. The current project settings needed for doing this can be found in the `storage/innobase/os/os0proj.c` source file.

- `innodb_autoinc_lock_mode`

The locking mode to use for generating auto-increment values. The allowable values are 0, 1, or 2, for “traditional”, “consecutive”, or “interleaved” lock mode, respectively. The characteristics of these modes are described in [Section 13.5.6.3, “How AUTO_INCREMENT Handling Works in InnoDB”](#).

This variable was added in MySQL 5.1.22 with a default of 1 (“consecutive” lock mode). Before 5.1.22, InnoDB uses “traditional” lock mode.

- `innodb_buffer_pool_size`

The size in bytes of the memory buffer InnoDB uses to cache data and indexes of its tables. The larger you set this value, the less disk I/O is needed to access data in tables. On a dedicated database server, you may set this to up to 80% of the machine physical memory size. However, do not set it too large because competition for physical memory might cause paging in the operating system.

- `innodb_checksums`

InnoDB can use checksum validation on all pages read from the disk to ensure extra fault tolerance against broken hardware or data files. This validation is enabled by default. However, under some rare circumstances (such as when running benchmarks) this extra safety feature is unneeded and can be disabled with `--skip-innodb-checksums`.

- `innodb_commit_concurrency`

The number of threads that can commit at the same time. Setting this parameter to 0 allows any number of transactions to commit simultaneously.

- `innodb_concurrency_tickets`

The number of threads that can enter InnoDB concurrently is determined by the `innodb_thread_concurrency` variable. A thread is placed in a queue when it tries to enter InnoDB if the number of threads has already reached the concurrency limit. When a thread is allowed to enter InnoDB, it is given a number of “free tickets” equal to the value of `innodb_concurrency_tickets`, and the thread can enter and leave InnoDB freely until it has used up its tickets. After that point, the thread again becomes subject to the concurrency check (and possible queuing) the next time it tries to enter InnoDB.

- `innodb_data_file_path`

The paths to individual data files and their sizes. The full directory path to each data file is formed by concatenating `innodb_data_home_dir` to each path specified here. The file sizes are specified in MB or GB (1024MB) by appending `M` or `G` to

the size value. The sum of the sizes of the files must be at least 10MB. If you do not specify `innodb_data_file_path`, the default behavior is to create a single 10MB auto-extending data file named `ibdata1`. The size limit of individual files is determined by your operating system. You can set the file size to more than 4GB on those operating systems that support big files. You can also use raw disk partitions as data files. See [Section 13.5.3.2, “Using Raw Devices for the Shared Tablespace”](#).

- `innodb_data_home_dir`

The common part of the directory path for all **InnoDB** data files. If you do not set this value, the default is the MySQL data directory. You can specify the value as an empty string, in which case you can use absolute file paths in `innodb_data_file_path`.

- `innodb_doublewrite`

By default, **InnoDB** stores all data twice, first to the doublewrite buffer, and then to the actual data files. This variable is enabled by default. It can be turned off with `--skip-innodb_doublewrite` for benchmarks or cases when top performance is needed rather than concern for data integrity or possible failures.

- `innodb_fast_shutdown`

If you set this variable to 0, **InnoDB** does a full purge and an insert buffer merge before a shutdown. These operations can take minutes, or even hours in extreme cases. If you set this variable to 1, **InnoDB** skips these operations at shutdown. The default value is 1. If you set it to 2, **InnoDB** will just flush its logs and then shut down cold, as if MySQL had crashed; no committed transaction will be lost, but crash recovery will be done at the next startup. A value of 2 cannot be used on NetWare.

- `innodb_file_io_threads`

The number of file I/O threads in **InnoDB**. Normally, this should be left at the default value of 4, but disk I/O on Windows may benefit from a larger number. On Unix, increasing the number has no effect; **InnoDB** always uses the default value.

- `innodb_file_per_table`

If this variable is enabled, **InnoDB** creates each new table using its own `.ibd` file for storing data and indexes, rather than in the shared tablespace. The default is to create tables in the shared tablespace. See [Section 13.5.3.1, “Using Per-Table Tablespaces”](#).

- `innodb_flush_log_at_trx_commit`

When `innodb_flush_log_at_trx_commit` is set to 0, the log buffer is written out to the log file once per second and the flush to disk operation is performed on the log file, but nothing is done at a transaction commit. When this value is 1 (the default), the log buffer is written out to the log file at each transaction commit and the flush to disk operation is performed on the log file. When set to 2, the log buffer is written out to the file at each commit, but the flush to disk operation is not performed on it. However, the flushing on the log file takes place once per second also when the value is 2. Note that the once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues.

The default value of this variable is 1, which is the value that is required for ACID compliance. You can achieve better performance by setting the value different from 1, but then you can lose at most one second worth of transactions in a crash. If you set the value to 0, then any `mysqld` process crash can erase the last second of transactions. If you set the value to 2, then only an operating system crash or a power outage can erase the last second of transactions. However, **InnoDB**'s crash recovery is not affected and thus crash recovery does work regardless of the value. Note that many operating systems and some disk hardware fool the flush-to-disk operation. They may tell `mysqld` that the flush has taken place, even though it has not. Then the durability of transactions is not guaranteed even with the setting 1, and in the worst case a power outage can even corrupt the **InnoDB** database. Using a battery-backed disk cache in the SCSI disk controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try using the Unix command `hdparm` to disable the caching of disk writes in hardware caches, or use some other command specific to the hardware vendor.

Note: For the greatest possible durability and consistency in a replication setup using **InnoDB** with transactions, you should use `innodb_flush_log_at_trx_commit=1` and `sync_binlog=1` in your master server `my.cnf` file.

- `innodb_flush_method`

If set to `fdatasync` (the default), **InnoDB** uses `fsync()` to flush both the data and log files. If set to `O_DSYNC`, **InnoDB** uses `O_SYNC` to open and flush the log files, but uses `fsync()` to flush the data files. If `O_DIRECT` is specified (available on some GNU/Linux versions, FreeBSD and Solaris), **InnoDB** uses `O_DIRECT` (or `directio()` on Solaris) to open the data files, and uses `fsync()` to flush both the data and log files. Note that **InnoDB** uses `fsync()` instead of `fdatasync()`, and it does not use `O_DSYNC` by default because there have been problems with it on many varieties of Unix. This variable is relevant only for Unix. On Windows, the flush method is always `async_unbuffered` and cannot be changed.

Different values of this variable can have a marked effect on **InnoDB performance**. For example, on some systems where **In-**

`noDB` data and log files are located on a SAN, it has been found that setting `innodb_flush_method` to `O_DIRECT` can degrade performance of simple `SELECT` statements by a factor of three.

- `innodb_force_recovery`

The crash recovery mode. Warning: This variable should be set greater than 0 only in an emergency situation when you want to dump your tables from a corrupt database! Possible values are from 1 to 6. The meanings of these values are described in [Section 13.5.8.1, “Forcing InnoDB Recovery”](#). As a safety measure, `InnoDB` prevents any changes to its data when this variable is greater than 0.

- `innodb_lock_wait_timeout`

The timeout in seconds an `InnoDB` transaction may wait for a lock before being rolled back. `InnoDB` automatically detects transaction deadlocks in its own lock table and rolls back the transaction. `InnoDB` notices locks set using the `LOCK TABLES` statement. The default is 50 seconds.

- `innodb_locks_unsafe_for_binlog`

This variable controls next-key locking in `InnoDB` searches and index scans. By default, this variable is 0 (disabled), which means that next-key locking is enabled.

Normally, `InnoDB` uses an algorithm called *next-key locking*. `InnoDB` performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on any index records it encounters. Thus, the row-level locks are actually index record locks. The locks that `InnoDB` sets on index records also affect the “gap” preceding that index record. If a user has a shared or exclusive lock on record *R* in an index, another user cannot insert a new index record immediately before *R* in the order of the index. Enabling this variable causes `InnoDB` not to use next-key locking in searches or index scans. Next-key locking is still used to ensure foreign key constraints and duplicate key checking. Note that enabling this variable may cause phantom problems: Suppose that you want to read and lock all children from the `child` table with an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

Suppose that there is an index on the `id` column. The query scans that index starting from the first record where `id` is greater than 100. If the locks set on the index records do not lock out inserts made in the gaps, another client can insert a new row into the table. If you execute the same `SELECT` within the same transaction, you see a new row in the result set returned by the query. This also means that if new items are added to the database, `InnoDB` does not guarantee serializability. Therefore, if this variable is enabled, `InnoDB` guarantees at most isolation level `READ COMMITTED`. (Conflict serializability is still guaranteed.)

Enabling this variable has an additional effect: `InnoDB` in an `UPDATE` or a `DELETE` only locks rows that it updates or deletes. This greatly reduces the probability of deadlocks, but they can happen. Note that enabling this variable still does not allow operations such as `UPDATE` to overtake other similar operations (such as another `UPDATE`) even in the case when they affect different rows. Consider the following example, beginning with this table:

```
CREATE TABLE A(A INT NOT NULL, B INT) ENGINE = InnoDB;
INSERT INTO A VALUES (1,2), (2,3), (3,2), (4,3), (5,2);
COMMIT;
```

Suppose that one client executes these statements:

```
SET AUTOCOMMIT = 0;
UPDATE A SET B = 5 WHERE B = 3;
```

Then suppose that another client executes these statements following those of the first client:

```
SET AUTOCOMMIT = 0;
UPDATE A SET B = 4 WHERE B = 2;
```

In this case, the second `UPDATE` must wait for a commit or rollback of the first `UPDATE`. The first `UPDATE` has an exclusive lock on row (2,3), and the second `UPDATE` while scanning rows also tries to acquire an exclusive lock for the same row, which it cannot have. This is because `UPDATE` first acquires an exclusive lock on a row and then determines whether the row belongs to the result set. If not, it releases the unnecessary lock, when the `innodb_locks_unsafe_for_binlog` variable is enabled.

Therefore, `InnoDB` executes `UPDATE` one as follows:

```
x-lock(1,2)
unlock(1,2)
```



```
x-lock(2,3)
update(2,3) to (2,5)
x-lock(3,2)
unlock(3,2)
x-lock(4,3)
update(4,3) to (4,5)
x-lock(5,2)
unlock(5,2)
```

InnoDB executes **UPDATE** two as follows:

```
x-lock(1,2)
update(1,2) to (1,4)
x-lock(2,3) - wait for query one to commit or rollback
```

- **innodb_log_archive**

Whether to log InnoDB archive files. This variable is present for historical reasons, but is unused. Recovery from a backup is done by MySQL using its own log files, so there is no need to archive InnoDB log files. The default for this variable is 0.

- **innodb_log_buffer_size**

The size in bytes of the buffer that InnoDB uses to write to the log files on disk. Sensible values range from 1MB to 8MB. The default is 1MB. A large log buffer allows large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have big transactions, making the log buffer larger saves disk I/O.

- **innodb_log_file_size**

The size in bytes of each log file in a log group. The combined size of log files must be less than 4GB on 32-bit computers. The default is 5MB. Sensible values range from 1MB to $1/N$ -th of the size of the buffer pool, where N is the number of log files in the group. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. But larger log files also mean that recovery is slower in case of a crash.

- **innodb_log_files_in_group**

The number of log files in the log group. InnoDB writes to the files in a circular fashion. The default (and recommended) is 2.

- **innodb_log_group_home_dir**

The directory path to the InnoDB log files. If you do not specify any InnoDB log variables, the default is to create two 5MB files names `ib_logfile0` and `ib_logfile1` in the MySQL data directory.

- **innodb_max_dirty_pages_pct**

This is an integer in the range from 0 to 100. The default is 90. The main thread in InnoDB tries to write pages from the buffer pool so that the percentage of dirty (not yet written) pages will not exceed this value.

- **innodb_max_purge_lag**

This variable controls how to delay **INSERT**, **UPDATE** and **DELETE** operations when the purge operations are lagging (see [Section 13.5.12, “Implementation of Multi-Versioning”](#)). The default value of this variable is 0, meaning that there are no delays.

The InnoDB transaction system maintains a list of transactions that have delete-marked index records by **UPDATE** or **DELETE** operations. Let the length of this list be *purge_lag*. When *purge_lag* exceeds *innodb_max_purge_lag*, each **INSERT**, **UPDATE** and **DELETE** operation is delayed by $((\text{purge\_lag}/\text{innodb\_max\_purge\_lag}) \times 10) - 5$ milliseconds. The delay is computed in the beginning of a purge batch, every ten seconds. The operations are not delayed if purge cannot run because of an old consistent read view that could see the rows to be purged.

A typical setting for a problematic workload might be 1 million, assuming that our transactions are small, only 100 bytes in size, and we can allow 100MB of unpurged rows in our tables.

- **innodb_mirrored_log_groups**

The number of identical copies of log groups to keep for the database. Currently, this should be set to 1.

- **innodb_open_files**

This variable is relevant only if you use multiple tablespaces in **InnoDB**. It specifies the maximum number of **.ibd** files that **InnoDB** can keep open at one time. The minimum value is 10. The default is 300.

The file descriptors used for **.ibd** files are for **InnoDB** only. They are independent of those specified by the **-open-files-limit** server option, and do not affect the operation of the table cache.

- **innodb_rollback_on_timeout**

In MySQL 5.1, **InnoDB** rolls back only the last statement on a transaction timeout. If this option is given, a transaction timeout causes **InnoDB** to abort and roll back the entire transaction (the same behavior as in MySQL 4.1). This variable was added in MySQL 5.1.15.

- **innodb_stats_on_metadata**

When this variable is enabled (which is the default, as before the variable was created), **InnoDB** updates statistics during metadata statements such as **SHOW TABLE STATUS** or **SHOW INDEX**, or when accessing the **INFORMATION_SCHEMA** tables **TABLES** or **STATISTICS**. (These updates are similar to what happens for **ANALYZE TABLE**.) When disabled, **InnoDB** does not update statistics during these operations. Disabling this variable can improve access speed for schemas that have a large number of tables or indexes. It can also improve the stability of execution plans for queries that involve **InnoDB** tables.

This variable was added in MySQL 5.1.17.

- **innodb_support_xa**

When set to **ON** or 1 (the default), this variable enables **InnoDB** support for two-phase commit in XA transactions. Enabling **innodb_support_xa** causes an extra disk flush for transaction preparation.

If you don't care about using XA, you can disable this variable by setting it to **OFF** or 0 to reduce the number of disk flushes and get better **InnoDB** performance. If you are using replication and/or the binary log then you should set this variable to **ON** or 1 to ensure that the binary log does not get out of sync compared to the table data.

- **innodb_sync_spin_loops**

The number of times a thread waits for an **InnoDB** mutex to be freed before the thread is suspended.

- **innodb_table_locks**

If **AUTOCOMMIT=0**, **InnoDB** honors **LOCK TABLES**; MySQL does not return from **LOCK TABLE ... WRITE** until all other threads have released all their locks to the table. The default value of **innodb_table_locks** is 1, which means that **LOCK TABLES** causes **InnoDB** to lock a table internally if **AUTOCOMMIT=0**.

- **innodb_thread_concurrency**

InnoDB tries to keep the number of operating system threads concurrently inside **InnoDB** less than or equal to the limit given by this variable. Once the number of threads reaches this limit, additional threads are placed into a wait state within a FIFO queue for execution. Threads waiting for locks are not counted in the number of concurrently executing threads.

The correct value for this variable is dependent on environment and workload. You will need to try a range of different values to determine what value works for your application.

The range of this variable is 0 to 1000. You can disable thread concurrency checking by setting the value to 0, which allows **InnoDB** to create as many threads as it needs.

The default value is 20 before MySQL 5.1.11, and 8 from 5.1.11 on.

- **innodb_thread_sleep_delay**

How long **InnoDB** threads sleep before joining the **InnoDB** queue, in microseconds. The default value is 10,000. A value of 0 disables sleep.

- **sync_binlog**

If the value of this variable is positive, the MySQL server synchronizes its binary log to disk (**fdatasync()**) after every **sync_binlog** writes to this binary log. Note that there is one write to the binary log per statement if in autocommit mode, and otherwise one write per transaction. The default value is 0 which does no synchronizing to disk. A value of 1 is the safest choice, be-

cause in the event of a crash you lose at most one statement/transaction from the binary log; however, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

13.5.5. Creating the InnoDB Tablespace

Suppose that you have installed MySQL and have edited your option file so that it contains the necessary InnoDB configuration parameters. Before starting MySQL, you should verify that the directories you have specified for InnoDB data files and log files exist and that the MySQL server has access rights to those directories. InnoDB does not create directories, only files. Check also that you have enough disk space for the data and log files.

It is best to run the MySQL server `mysqld` from the command prompt when you first start the server with InnoDB enabled, not from the `mysqld_safe` wrapper or as a Windows service. When you run from a command prompt you see what `mysqld` prints and what is happening. On Unix, just invoke `mysqld`. On Windows, use the `--console` option.

When you start the MySQL server after initially configuring InnoDB in your option file, InnoDB creates your data files and log files, and prints something like this:

```
InnoDB: The first specified datafile /home/heikki/data/ibdata1
did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file /home/heikki/data/ibdata1 size to 134217728
InnoDB: Database physically writes the file full: wait...
InnoDB: datafile /home/heikki/data/ibdata2 did not exist:
new to be created
InnoDB: Setting file /home/heikki/data/ibdata2 size to 262144000
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file /home/heikki/data/logs/ib_logfile0 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile0 size
to 5242880
InnoDB: Log file /home/heikki/data/logs/ib_logfile1 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile1 size
to 5242880
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
InnoDB: Started
mysqld: ready for connections
```

At this point InnoDB has initialized its tablespace and log files. You can connect to the MySQL server with the usual MySQL client programs like `mysql`. When you shut down the MySQL server with `mysqladmin shutdown`, the output is like this:

```
010321 18:33:34 mysqld: Normal shutdown
010321 18:33:34 mysqld: Shutdown Complete
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
```

You can look at the data file and log directories and you see the files created there. When MySQL is started again, the data files and log files have been created already, so the output is much briefer:

```
InnoDB: Started
mysqld: ready for connections
```

If you add the `innodb_file_per_table` option to `my.cnf`, InnoDB stores each table in its own `.ibd` file in the same MySQL database directory where the `.frm` file is created. See [Section 13.5.3.1, “Using Per-Table Tablespaces”](#).

13.5.5.1. Dealing with InnoDB Initialization Problems

If InnoDB prints an operating system error during a file operation, usually the problem has one of the following causes:

- You did not create the InnoDB data file directory or the InnoDB log directory.
- `mysqld` does not have access rights to create files in those directories.
- `mysqld` cannot read the proper `my.cnf` or `my.ini` option file, and consequently does not see the options that you specified.
- The disk is full or a disk quota is exceeded.

- You have created a subdirectory whose name is equal to a data file that you specified, so the name cannot be used as a filename.
- There is a syntax error in the `innodb_data_home_dir` or `innodb_data_file_path` value.

If something goes wrong when **InnoDB** attempts to initialize its tablespace or its log files, you should delete all files created by **InnoDB**. This means all `ibdata` files and all `ib_logfile` files. In case you have already created some **InnoDB** tables, delete the corresponding `.frm` files for these tables (and any `.ibd` files if you are using multiple tablespaces) from the MySQL database directories as well. Then you can try the **InnoDB** database creation again. It is best to start the MySQL server from a command prompt so that you see what is happening.

13.5.6. Creating and Using **InnoDB** Tables

To create an **InnoDB** table, specify an `ENGINE = InnoDB` option in the `CREATE TABLE` statement:

```
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB;
```

The statement creates a table and an index on column `a` in the **InnoDB** tablespace that consists of the data files that you specified in `my.cnf`. In addition, MySQL creates a file `customers.frm` in the `test` directory under the MySQL database directory. Internally, **InnoDB** adds an entry for the table to its own data dictionary. The entry includes the database name. For example, if `test` is the database in which the `customers` table is created, the entry is for `'test/customers'`. This means you can create a table of the same name `customers` in some other database, and the table names do not collide inside **InnoDB**.

You can query the amount of free space in the **InnoDB** tablespace by issuing a `SHOW TABLE STATUS` statement for any **InnoDB** table. The amount of free space in the tablespace appears in the `Comment` section in the output of `SHOW TABLE STATUS`. For example:

```
SHOW TABLE STATUS FROM test LIKE 'customers'
```

Note that the statistics `SHOW` displays for **InnoDB** tables are only approximate. They are used in SQL optimization. Table and index reserved sizes in bytes are accurate, though.

13.5.6.1. How to Use Transactions in **InnoDB** with Different APIs

By default, each client that connects to the MySQL server begins with autocommit mode enabled, which automatically commits every SQL statement as you execute it. To use multiple-statement transactions, you can switch autocommit off with the SQL statement `SET AUTOCOMMIT = 0` and use `COMMIT` and `ROLLBACK` to commit or roll back your transaction. If you want to leave autocommit on, you can enclose your transactions within `START TRANSACTION` and either `COMMIT` or `ROLLBACK`. The following example shows two transactions. The first is committed; the second is rolled back.

```
shell> mysql test

mysql> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A))
-> ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM CUSTOMER;
+-----+-----+
| A     | B     |
+-----+-----+
| 10    | Heikki |
+-----+-----+
1 row in set (0.00 sec)
mysql>
```

In APIs such as PHP, Perl DBI, JDBC, ODBC, or the standard C call interface of MySQL, you can send transaction control statements such as `COMMIT` to the MySQL server as strings just like any other SQL statements such as `SELECT` or `INSERT`. Some APIs also offer separate special transaction commit and rollback functions or methods.

13.5.6.2. Converting MyISAM Tables to InnoDB

Important: Do not convert MySQL system tables in the `mysql` database (such as `user` or `host`) to the `InnoDB` type. This is an unsupported operation. The system tables must always be of the `MyISAM` type.

If you want all your (non-system) tables to be created as `InnoDB` tables, you can simply add the line `default-storage-engine=innodb` to the `[mysqld]` section of your server option file.

`InnoDB` does not have a special optimization for separate index creation the way the `MyISAM` storage engine does. Therefore, it does not pay to export and import the table and create indexes afterward. The fastest way to alter a table to `InnoDB` is to do the inserts directly to an `InnoDB` table. That is, use `ALTER TABLE ... ENGINE=INNODB`, or create an empty `InnoDB` table with identical definitions and insert the rows with `INSERT INTO ... SELECT * FROM ...`.

If you have `UNIQUE` constraints on secondary keys, you can speed up a table import by turning off the uniqueness checks temporarily during the import operation:

```
SET UNIQUE_CHECKS=0;
... import operation ...
SET UNIQUE_CHECKS=1;
```

For big tables, this saves a lot of disk I/O because `InnoDB` can then use its insert buffer to write secondary index records as a batch. Be certain that the data contains no duplicate keys. `UNIQUE_CHECKS` allows but does not require storage engines to ignore duplicate keys.

To get better control over the insertion process, it might be good to insert big tables in pieces:

```
INSERT INTO newtable SELECT * FROM oldtable
WHERE yourkey > something AND yourkey <= somethingelse;
```

After all records have been inserted, you can rename the tables.

During the conversion of big tables, you should increase the size of the `InnoDB` buffer pool to reduce disk I/O. Do not use more than 80% of the physical memory, though. You can also increase the sizes of the `InnoDB` log files.

Make sure that you do not fill up the tablespace: `InnoDB` tables require a lot more disk space than `MyISAM` tables. If an `ALTER TABLE` operation runs out of space, it starts a rollback, and that can take hours if it is disk-bound. For inserts, `InnoDB` uses the insert buffer to merge secondary index records to indexes in batches. That saves a lot of disk I/O. For rollback, no such mechanism is used, and the rollback can take 30 times longer than the insertion.

In the case of a runaway rollback, if you do not have valuable data in your database, it may be advisable to kill the database process rather than wait for millions of disk I/O operations to complete. For the complete procedure, see [Section 13.5.8.1, “Forcing InnoDB Recovery”](#).

13.5.6.3. How AUTO_INCREMENT Handling Works in InnoDB

Beginning with MySQL 5.1.22, `InnoDB` provides a locking strategy that significantly improves scalability and performance of SQL statements that add rows to tables with `AUTO_INCREMENT` columns. This section provides background information on the original (“traditional”) implementation of auto-increment locking in `InnoDB`, explains the configurable locking mechanism, documents the parameter for configuring the mechanism, and describes its behavior and interaction with replication.

13.5.6.3.1. “Traditional” InnoDB Auto-Increment Locking

The original implementation of auto-increment handling in `InnoDB` uses the following strategy to prevent problems when using the binary log for statement-based replication or for certain recovery scenarios.

If you specify an `AUTO_INCREMENT` column for an `InnoDB` table, the table handle in the `InnoDB` data dictionary contains a special counter called the auto-increment counter that is used in assigning new values for the column. This counter is stored only in main memory, not on disk.

`InnoDB` uses the following algorithm to initialize the auto-increment counter for a table `t` that contains an `AUTO_INCREMENT` column named `ai_col`: After a server startup, for the first insert into a table `t`, `InnoDB` executes the equivalent of this statement:

```
SELECT MAX(ai_col) FROM t FOR UPDATE;
```

`InnoDB` increments by one the value retrieved by the statement and assigns it to the column and to the auto-increment counter for the

table. If the table is empty, **InnoDB** uses the value **1**. If a user invokes a **SHOW TABLE STATUS** statement that displays output for the table **t** and the auto-increment counter has not been initialized, **InnoDB** initializes but does not increment the value and stores it for use by later inserts. This initialization uses a normal exclusive-locking read on the table and the lock lasts to the end of the transaction.

InnoDB follows the same procedure for initializing the auto-increment counter for a freshly created table.

After the auto-increment counter has been initialized, if a user does not explicitly specify a value for an **AUTO_INCREMENT** column, **InnoDB** increments the counter by one and assigns the new value to the column. If the user inserts a row that explicitly specifies the column value, and the value is bigger than the current counter value, the counter is set to the specified column value.

When accessing the auto-increment counter, **InnoDB** uses a special table-level **AUTO-INC** lock that it keeps to the end of the current SQL statement, not to the end of the transaction. The special lock release strategy was introduced to improve concurrency for inserts into a table containing an **AUTO_INCREMENT** column. Nevertheless, two transactions cannot have the **AUTO-INC** lock on the same table simultaneously, which can have a performance impact if the **AUTO-INC** lock is held for a long time. That might be the case for a statement such as **INSERT INTO t1 ... SELECT ... FROM t2** that inserts all rows from one table into another.

InnoDB uses the in-memory auto-increment counter as long as the server runs. When the server is stopped and restarted, **InnoDB** reinitializes the counter for each table for the first **INSERT** to the table, as described earlier.

You may see gaps in the sequence of values assigned to the **AUTO_INCREMENT** column if you roll back transactions that have generated numbers using the counter.

If a user specifies **NULL** or **0** for the **AUTO_INCREMENT** column in an **INSERT**, **InnoDB** treats the row as if the value had not been specified and generates a new value for it.

The behavior of the auto-increment mechanism is not defined if a user assigns a negative value to the column or if the value becomes bigger than the maximum integer that can be stored in the specified integer type.

An **AUTO_INCREMENT** column must be the first column listed if it is part of a multiple-column index in an **InnoDB** table.

InnoDB supports the **AUTO_INCREMENT = N** table option in **CREATE TABLE** and **ALTER TABLE** statements, to set the initial counter value or alter the current counter value. The effect of this option is canceled by a server restart, for reasons discussed earlier in this section.

13.5.6.3.2. Configurable **InnoDB** Auto-Increment Locking

As described in the previous section, **InnoDB** uses a special lock called the **AUTO-INC** table-level lock for inserts into tables with **AUTO_INCREMENT** columns. This lock is normally held to the end of the statement (not to the end of the transaction), to ensure that auto-increment numbers are assigned in a predictable and repeatable order for a given sequence of **INSERT** statements.

In the case of statement-based replication, this means that when a SQL statement is replicated on a slave server, the same values are used for the auto-increment column as on the master server. The result of execution of multiple **INSERT** statements is deterministic, and the slave reproduces the same data as on the master. If auto-increment values generated by multiple **INSERT** statements were interleaved, the result of two concurrent **INSERT** statements would be non-deterministic, and could not reliably be propagated to a slave server using statement-based replication.

To make this clear, consider an example that uses this table:

```
CREATE TABLE t1 (
  c1 INT(11) NOT NULL AUTO_INCREMENT,
  c2 VARCHAR(10) DEFAULT NULL,
  PRIMARY KEY (c1)
) ENGINE=InnoDB;
```

Suppose that there are two transactions running, each inserting rows into a table with an **AUTO_INCREMENT** column. One transaction is using an **INSERT ... SELECT** statement that inserts 1000 rows, and another is using a simple **INSERT** statement that inserts one row:

```
Tx1: INSERT INTO t1 (c2) SELECT 1000 rows from another table ...
Tx2: INSERT INTO t1 (c2) VALUES ('xxx');
```

InnoDB cannot tell in advance how many rows will be retrieved from the **SELECT** in the **INSERT** statement in Tx1, and it assigns the auto-increment values one at a time as the statement proceeds. With a table-level lock, held to the end of the statement, only one **INSERT** statement referring to table **t1** can execute at a time, and the generation of auto-increment numbers by different statements is not interleaved. The auto-increment value generated by the Tx1 **INSERT ... SELECT** statement will be consecutive, and the (single) auto-increment value used by the **INSERT** statement in Tx2 will either be smaller or larger than all those used for Tx1, depending on which statement executes first.

As long as the SQL statements execute in the same order when replayed from the binary log (when using statement-based replication, or in recovery scenarios), the results will be the same as they were when Tx1 and Tx2 first ran. Thus, table-level locks held until the end of a statement make **INSERT** statements using auto-increment safe for use with statement-based replication. However, those locks limit concurrency and scalability when multiple transactions are executing insert statements at the same time.

In the preceding example, if there were no table-level lock, the value of the auto-increment column used for the **INSERT** in Tx2 depends on precisely when the statement executes. If the **INSERT** of Tx2 executes while the **INSERT** of Tx1 is running (rather than before it starts or after it completes), the specific auto-increment values assigned by the two **INSERT** statements are non-deterministic, and may vary from run to run.

As of MySQL 5.1.22, **InnoDB** can avoid using the table-level AUTO-INC lock for a class of **INSERT** statements where the number of rows is known in advance, and still preserve deterministic execution and safety for statement-based replication. Further, if you are not using the binary log to replay SQL statements as part of recovery or replication, you can entirely eliminate use of the AUTO-INC table-level lock for even greater concurrency and performance—at the cost of permitting gaps in auto-increment numbers assigned by a statement and potentially having the numbers assigned by concurrently executing statements interleaved.

For **INSERT** statements where the number of rows to be inserted is known at the beginning of processing the statement, **InnoDB** quickly allocates the required number of auto-increment values without taking any lock, but only if there is no concurrent session already holding the table-level lock AUTO-INC lock (because that other statement will be allocating auto-increment values one-by-one as it proceeds). More precisely, such an **INSERT** statement obtains auto-increment values under the control of a mutex (a light-weight lock) that is *not* held until the statement completes, but only for the duration of the allocation process.

This new locking scheme allows much greater scalability, but it does introduce some subtle differences in how auto-increment values are assigned compared to the original mechanism. To describe the way auto-increment works in **InnoDB**, the following discussion defines some terms, and explains how **InnoDB** behaves using different settings of the new `innodb_autoinc_lock_mode` configuration parameter. Additional considerations are described following the explanation of auto-increment locking behavior.

First, some definitions:

- “**INSERT-like**” statements

All statements that generate new rows in a table, including **INSERT**, **INSERT ... SELECT**, **REPLACE**, **REPLACE ... SELECT**, and **LOAD DATA**.

- “Simple inserts”

Statements for which the number of rows to be inserted can be determined in advance (when the statement is initially processed). This includes single-row and multiple-row **INSERT**, **INSERT ... ON DUPLICATE KEY UPDATE**, and **REPLACE** statements that do not have a nested subquery.

- “Bulk inserts”

Statements for which the number of rows to be inserted (and the number of required auto-increment values) is not known in advance. This includes **INSERT ... SELECT**, **REPLACE ... SELECT**, and **LOAD DATA** statements. **InnoDB** will assign new values for the **AUTO_INCREMENT** column one at a time as each row is processed.

- “Mixed-mode inserts”

These are “simple insert” statements that specify the auto-increment value for some (but not all) of the new rows. An example follows, where **c1** is an **AUTO_INCREMENT** column of table **t1**:

```
INSERT INTO t1 (c1,c2) VALUES (1, 'a'), (NULL, 'b'), (5, 'c'), (NULL, 'd');
```

Beginning with MySQL 5.1.22, there is a new configuration parameter that controls how **InnoDB** uses locking when generating values for **AUTO_INCREMENT** columns. This parameter can be set using the `--innodb-autoinc-lock-mode` option at `mysqld` startup, or at runtime by setting the global `innodb_autoinc_lock_mode` system variable. Although this variable can be changed at runtime, the behavior is undefined if you do so while transactions that generate auto-increment values are executing.

In general, if you encounter problems with the way auto-increment works (which will most likely involve replication), you can force use of the original behavior by setting the lock mode to 0.

There are three possible settings for the `innodb_autoinc_lock_mode` parameter:

- `innodb_autoinc_lock_mode = 0` (“traditional” lock mode)

This lock mode provides the same behavior as before `innodb_autoinc_lock_mode` existed. For all “`INSERT`-like” statements, a special AUTO-INC table-level lock is obtained and held to the end of the statement. This assures that the auto-increment values assigned by any given statement are consecutive (although “gaps” can exist within a table if a transaction that generated auto-increment values is rolled back, as discussed later).

This lock mode is provided only for backward compatibility and performance testing. There is little reason to use this lock mode unless you use “mixed-mode inserts” and care about the minor difference in semantics described later.

- `innodb_autoinc_lock_mode = 1` (“consecutive” lock mode)

This is the default lock mode. In this mode, “bulk inserts” use the special table-level AUTO-INC table-level lock and holds it until the end of the statement. This applies to all `INSERT ... SELECT`, `REPLACE ... SELECT`, and `LOAD DATA` statements. Only one statement holding the AUTO-INC lock can execute at a time.

With this lock mode, “simple inserts” (only) use a new locking model where a light-weight mutex is used during the allocation of auto-increment values, and no AUTO-INC table-level lock is used, unless an AUTO-INC lock is held by another transaction. If another transaction does hold an AUTO-INC lock, a “simple insert” waits for the AUTO-INC lock, as if it too were a “bulk insert.”

This lock mode ensures that, in the presence of `INSERT` statements where the number of rows is not known in advance (and where auto-increment numbers are assigned as the statement progresses), all auto-increment values assigned by any “`INSERT`-like” statement are consecutive, and operations are safe for statement-based replication.

Simply put, the important impact of this lock mode is significantly better scalability. This mode is safe for use with statement-based replication. Further, as with “traditional” lock mode, auto-increment numbers assigned by any given statement are *consecutive*. In this mode, there is *no change* in semantics compared to “traditional” mode for any statement that uses auto-increment, with one minor exception.

The exception is for “mixed-mode inserts”, where the user provides explicit values for an `AUTO_INCREMENT` column for some, but not all, rows in a multiple-row “simple insert.” For such inserts, `InnoDB` will allocate more auto-increment values than the number of rows to be inserted. However, all values automatically assigned are consecutively generated (and thus higher than) the auto-increment value generated by the most recently executed previous statement. “Excess” numbers are lost.

- `innodb_autoinc_lock_mode = 2` (“interleaved” lock mode)

In this lock mode, no “`INSERT`-like” statements use the table-level AUTO-INC lock, and multiple statements can execute at the same time. This is the fastest and most scalable lock mode, but it is *not safe* when using statement-based replication or recovery scenarios when SQL statements are replayed from the binary log.

In this lock mode, auto-increment values are guaranteed to be unique and monotonically increasing across all concurrently executing “`INSERT`-like” statements. However, because multiple statements can be generating numbers at the same time (that is, allocation of numbers is *interleaved* across statements), the values generated for the rows inserted by any given statement may not be consecutive.

If the only statements executing are “simple inserts” where the number of rows to be inserted is known ahead of time, there will be no gaps in the numbers generated for a single statement, except for “mixed-mode inserts.” However, when “bulk inserts” are executed, there may be gaps in the auto-increment values assigned by any given statement.

The auto-increment locking modes provided by `innodb_autoinc_lock_mode` have several usage implications:

- Using auto-increment with replication

If you are using statement-based replication, you should set `innodb_autoinc_lock_mode` to 0 or 1 and use the same value on the master and its slaves. Auto-increment values are not ensured to be the same on the slaves as on the master if you use `innodb_autoinc_lock_mode = 2` (“interleaved”) or configurations where the master and slaves do not use the same lock mode.

If you are using row-based replication, all of the auto-increment lock modes are safe. Row-based replication is not sensitive to the order of execution of the SQL statements.

- “Lost” auto-increment values and sequence gaps

In all lock modes (0, 1, and 2), if a transaction that generated auto-increment values rolls back, those auto-increment values are “lost.” Once a value is generated for an auto-increment column, it cannot be rolled back, whether or not the “`INSERT`-like” state-

ment is completed, and whether or not the containing transaction is rolled back. Such lost values are not reused. Thus, there may be gaps in the values stored in an `AUTO_INCREMENT` column of a table.

- Auto-increment values assigned by “mixed-mode inserts”

Consider a “mixed-mode insert,” where a “simple insert” specifies the auto-increment value for some (but not all) resulting rows. Such a statement will behave differently in lock modes 0, 1, and 2. For example, assume `c1` is an `AUTO_INCREMENT` column of table `t1`, and that the most recent automatically generated sequence number is 100. Consider the following “mixed-mode insert” statement:

```
INSERT INTO t1 (c1,c2) VALUES (1, 'a'), (NULL, 'b'), (5, 'c'), (NULL, 'd');
```

With `innodb_autoinc_lock_mode` set to 0 (“traditional”), the four new rows will be:

c1	c2
1	a
101	b
5	c
102	d

The next available auto-increment value will be 103 because the auto-increment values are allocated one at a time, not all at once at the beginning of statement execution. This result is true whether or not there are concurrently executing “`INSERT`-like” statements (of any type).

With `innodb_autoinc_lock_mode` set to 1 (“consecutive”), the four new rows will also be:

c1	c2
1	a
101	b
5	c
102	d

However, in this case, the next available auto-increment value will be 105, not 103 because four auto-increment values are allocated at the time the statement is processed, but only two are used. This result is true whether or not there are concurrently executing “`INSERT`-like” statements (of any type).

With `innodb_autoinc_lock_mode` set to mode 2 (“interleaved”), the four new rows will be:

c1	c2
1	a
x	b
5	c
y	d

The values of `x` and `y` will be unique and larger than any previously generated rows. However, the specific values of `x` and `y` will depend on the number of auto-increment values generated by concurrently executing statements.

Finally, consider the following statement, issued when the most-recently generated sequence number was the value 4:

```
INSERT INTO t1 (c1,c2) VALUES (1, 'a'), (NULL, 'b'), (5, 'c'), (NULL, 'd');
```

With any `innodb_autoinc_lock_mode` setting, this statement will generate a duplicate-key error 23000 (`Can't write; duplicate key in table`) because 5 will be allocated for the row `(NULL, 'b')` and insertion of the row `(5, 'c')` will fail.

- Gaps in auto-increment values for “bulk inserts”

With `innodb_autoinc_lock_mode` set to 0 (“traditional”) or 1 (“consecutive”), the auto-increment values generated by any given statement will be consecutive, without gaps, because the table-level AUTO-INC lock is held until the end of the statement, and only one such statement can execute at a time.

With `innodb_autoinc_lock_mode` set to 2 (“interleaved”), there may be gaps in the auto-increment values generated by “bulk inserts,” but only if there are concurrently executing “**INSERT**-like” statements.

13.5.6.4. FOREIGN KEY Constraints

InnoDB also supports foreign key constraints. The syntax for a foreign key constraint definition in **InnoDB** looks like this:

```
[CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
REFERENCES tbl_name (index_col_name, ...)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Foreign keys definitions are subject to the following conditions:

- Both tables must be **InnoDB** tables and they must not be **TEMPORARY** tables.
- Corresponding columns in the foreign key and the referenced key must have similar internal data types inside **InnoDB** so that they can be compared without a type conversion. *The size and sign of integer types must be the same.* The length of string types need not be the same. For non-binary (character) string columns, the character set and collation must be the same.
- In the referencing table, there must be an index where the foreign key columns are listed as the *first* columns in the same order. Such an index is created on the referencing table automatically if it does not exist.
- In the referenced table, there must be an index where the referenced columns are listed as the *first* columns in the same order.
- Index prefixes on foreign key columns are not supported. One consequence of this is that **BLOB** and **TEXT** columns cannot be included in a foreign key, because indexes on those columns must always include a prefix length.
- If the **CONSTRAINT *symbol*** clause is given, the *symbol* value must be unique in the database. If the clause is not given, **InnoDB** creates the name automatically.

InnoDB rejects any **INSERT** or **UPDATE** operation that attempts to create a foreign key value in a child table if there is no a matching candidate key value in the parent table. The action **InnoDB** takes for any **UPDATE** or **DELETE** operation that attempts to update or delete a candidate key value in the parent table that has some matching rows in the child table is dependent on the *referential action* specified using **ON UPDATE** and **ON DELETE** subclauses of the **FOREIGN KEY** clause. When the user attempts to delete or update a row from a parent table, and there are one or more matching rows in the child table, **InnoDB** supports five options regarding the action to be taken:

- **CASCADE**: Delete or update the row from the parent table and automatically delete or update the matching rows in the child table. Both **ON DELETE CASCADE** and **ON UPDATE CASCADE** are supported. Between two tables, you should not define several **ON UPDATE CASCADE** clauses that act on the same column in the parent table or in the child table.
- **SET NULL**: Delete or update the row from the parent table and set the foreign key column or columns in the child table to **NULL**. This is valid only if the foreign key columns do not have the **NOT NULL** qualifier specified. Both **ON DELETE SET NULL** and **ON UPDATE SET NULL** clauses are supported.

If you specify a **SET NULL** action, *make sure that you have not declared the columns in the child table as **NOT NULL**.*

- **NO ACTION**: In standard SQL, **NO ACTION** means *no action* in the sense that an attempt to delete or update a primary key value is not allowed to proceed if there is a related foreign key value in the referenced table. **InnoDB** rejects the delete or update operation for the parent table.
- **RESTRICT**: Rejects the delete or update operation for the parent table. **NO ACTION** and **RESTRICT** are the same as omitting the **ON DELETE** or **ON UPDATE** clause. (Some database systems have deferred checks, and **NO ACTION** is a deferred check. In MySQL, foreign key constraints are checked immediately, so **NO ACTION** and **RESTRICT** are the same.)
- **SET DEFAULT**: This action is recognized by the parser, but **InnoDB** rejects table definitions containing **ON DELETE SET DEFAULT** or **ON UPDATE SET DEFAULT** clauses.

Note that **InnoDB** supports foreign key references within a table. In these cases, “child table records” really refers to dependent records

within the same table.

InnoDB requires indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. The index on the foreign key is created automatically. This is in contrast to some older versions, in which indexes had to be created explicitly or the creation of foreign key constraints would fail.

If MySQL reports an error number 1005 from a **CREATE TABLE** statement, and the error message refers to `errno 150`, table creation failed because a foreign key constraint was not correctly formed. Similarly, if an **ALTER TABLE** fails and it refers to `errno 150`, that means a foreign key definition would be incorrectly formed for the altered table. You can use **SHOW ENGINE INNODB STATUS** to display a detailed explanation of the most recent **InnoDB** foreign key error in the server.

Note

InnoDB does not check foreign key constraints on those foreign key or referenced key values that contain a **NULL** column.

Note

Currently, triggers are not activated by cascaded foreign key actions.

You cannot create a table with a column name that matches the name of an internal **InnoDB** column (including **DB_ROW_ID**, **DB_TRX_ID**, **DB_ROLL_PTR** and **DB_MIX_ID**). In versions of MySQL before 5.1.10 this would cause a crash, since 5.1.10 the server will report error 1005 and refers to `errno -1` in the error message.

Deviation from SQL standards: If there are several rows in the parent table that have the same referenced key value, **InnoDB** acts in foreign key checks as if the other parent rows with the same key value do not exist. For example, if you have defined a **RESTRICT** type constraint, and there is a child row with several parent rows, **InnoDB** does not allow the deletion of any of those parent rows.

InnoDB performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.

Deviation from SQL standards: A **FOREIGN KEY** constraint that references a non-**UNIQUE** key is not standard SQL. It is an **InnoDB** extension to standard SQL.

Deviation from SQL standards: If **ON UPDATE CASCADE** or **ON UPDATE SET NULL** recurses to update the *same table* it has previously updated during the cascade, it acts like **RESTRICT**. This means that you cannot use self-referential **ON UPDATE CASCADE** or **ON UPDATE SET NULL** operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential **ON DELETE SET NULL**, on the other hand, is possible, as is a self-referential **ON DELETE CASCADE**. Cascading operations may not be nested more than 15 levels deep.

Deviation from SQL standards: Like MySQL in general, in an SQL statement that inserts, deletes, or updates many rows, **InnoDB** checks **UNIQUE** and **FOREIGN KEY** constraints row-by-row. According to the SQL standard, the default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. Until **InnoDB** implements deferred constraint checking, some things will be impossible, such as deleting a record that refers to itself via a foreign key.

Here is a simple example that relates **parent** and **child** tables through a single-column foreign key:

```
CREATE TABLE parent (id INT NOT NULL,
                     PRIMARY KEY (id)
) ENGINE=INNODB;
CREATE TABLE child (id INT, parent_id INT,
                    INDEX par_ind (parent_id),
                    FOREIGN KEY (parent_id) REFERENCES parent(id)
                    ON DELETE CASCADE
) ENGINE=INNODB;
```

A more complex example in which a **product_order** table has foreign keys for two other tables. One foreign key references a two-column index in the **product** table. The other references a single-column index in the **customer** table:

```
CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,
                      price DECIMAL,
                      PRIMARY KEY(category, id)) ENGINE=INNODB;
CREATE TABLE customer (id INT NOT NULL,
                       PRIMARY KEY (id)) ENGINE=INNODB;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
                             product_category INT NOT NULL,
                             product_id INT NOT NULL,
                             customer_id INT NOT NULL,
                             PRIMARY KEY(no),
                             INDEX (product_category, product_id),
                             FOREIGN KEY (product_category, product_id)
```

```
REFERENCES product(category, id)
ON UPDATE CASCADE ON DELETE RESTRICT,
INDEX (customer_id),
FOREIGN KEY (customer_id)
REFERENCES customer(id)) ENGINE=INNODB;
```

InnoDB allows you to add a new foreign key constraint to a table by using **ALTER TABLE**:

```
ALTER TABLE tbl_name
ADD [CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
REFERENCES tbl_name (index_col_name, ...)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Remember to create the required indexes first. You can also add a self-referential foreign key constraint to a table using **ALTER TABLE**.

InnoDB also supports the use of **ALTER TABLE** to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

If the **FOREIGN KEY** clause included a **CONSTRAINT** name when you created the foreign key, you can refer to that name to drop the foreign key. Otherwise, the *fk_symbol* value is internally generated by **InnoDB** when the foreign key is created. To find out the symbol value when you want to drop a foreign key, use the **SHOW CREATE TABLE** statement. For example:

```
mysql> SHOW CREATE TABLE ibtest11c\G
***** 1. row *****
      Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default '',
  `C` varchar(175) default NULL,
  PRIMARY KEY (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`,`D`)
REFERENCES `ibtest11a` (`A`,`D`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`,`C`)
REFERENCES `ibtest11a` (`B`,`C`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB CHARSET=latin1
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY `0_38775`;
```

You cannot add a foreign key and drop a foreign key in separate clauses of a single **ALTER TABLE** statement. Separate statements are required.

If **ALTER TABLE** for an **InnoDB** table results in changes to column values (for example, because a column is truncated), **InnoDB**'s **FOREIGN KEY** constraint checks do not notice possible violations caused by changing the values.

The **InnoDB** parser allows table and column identifiers in a **FOREIGN KEY ... REFERENCES ...** clause to be quoted within backticks. (Alternatively, double quotes can be used if the **ANSI_QUOTES** SQL mode is enabled.) The **InnoDB** parser also takes into account the setting of the **lower_case_table_names** system variable.

InnoDB returns a table's foreign key definitions as part of the output of the **SHOW CREATE TABLE** statement:

```
SHOW CREATE TABLE tbl_name;
```

mysqldump also produces correct definitions of tables to the dump file, and does not forget about the foreign keys.

You can also display the foreign key constraints for a table like this:

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
```

The foreign key constraints are listed in the **Comment** column of the output.

When performing foreign key checks, **InnoDB** sets shared row-level locks on child or parent records it has to look at. **InnoDB** checks foreign key constraints immediately; the check is not deferred to transaction commit.

To make it easier to reload dump files for tables that have foreign key relationships, `mysqldump` automatically includes a statement in the dump output to set `FOREIGN_KEY_CHECKS` to 0. This avoids problems with tables having to be reloaded in a particular order when the dump is reloaded. It is also possible to set this variable manually:

```
mysql> SET FOREIGN_KEY_CHECKS = 0;
mysql> SOURCE dump_file_name;
mysql> SET FOREIGN_KEY_CHECKS = 1;
```

This allows you to import the tables in any order if the dump file contains tables that are not correctly ordered for foreign keys. It also speeds up the import operation. Setting `FOREIGN_KEY_CHECKS` to 0 can also be useful for ignoring foreign key constraints during `LOAD DATA` and `ALTER TABLE` operations. However, even if `FOREIGN_KEY_CHECKS=0`, InnoDB does not allow the creation of a foreign key constraint where a column references a non-matching column type. Also, if an InnoDB table has foreign key constraints, `ALTER TABLE` cannot be used to change the table to use another storage engine. To alter the storage engine, you must drop any foreign key constraints first.

InnoDB does not allow you to drop a table that is referenced by a `FOREIGN KEY` constraint, unless you do `SET FOREIGN_KEY_CHECKS=0`. When you drop a table, the constraints that were defined in its create statement are also dropped.

If you re-create a table that was dropped, it must have a definition that conforms to the foreign key constraints referencing it. It must have the right column names and types, and it must have indexes on the referenced keys, as stated earlier. If these are not satisfied, MySQL returns error number 1005 and refers to errno 150 in the error message.

13.5.6.5. InnoDB and MySQL Replication

MySQL replication works for InnoDB tables as it does for MyISAM tables. It is also possible to use replication in a way where the storage engine on the slave is not the same as the original storage engine on the master. For example, you can replicate modifications to an InnoDB table on the master to a MyISAM table on the slave.

To set up a new slave for a master, you have to make a copy of the InnoDB tablespace and the log files, as well as the `.frm` files of the InnoDB tables, and move the copies to the slave. If the `innodb_file_per_table` variable is enabled, you must also copy the `.ibd` files as well. For the proper procedure to do this, see [Section 13.5.8, “Backing Up and Recovering an InnoDB Database”](#).

If you can shut down the master or an existing slave, you can take a cold backup of the InnoDB tablespace and log files and use that to set up a slave. To make a new slave without taking down any server you can also use the non-free (commercial) [InnoDB Hot Backup tool](#).

You cannot set up replication for InnoDB using the `LOAD TABLE FROM MASTER` statement, which works only for MyISAM tables. There are two possible workarounds:

- Dump the table on the master and import the dump file into the slave.
- Use `ALTER TABLE tbl_name ENGINE=MyISAM` on the master before setting up replication with `LOAD TABLE tbl_name FROM MASTER`, and then use `ALTER TABLE` to convert the master table back to InnoDB afterward. However, this should not be done for tables that have foreign key definitions because the definitions will be lost.

Transactions that fail on the master do not affect replication at all. MySQL replication is based on the binary log where MySQL writes SQL statements that modify data. A transaction that fails (for example, because of a foreign key violation, or because it is rolled back) is not written to the binary log, so it is not sent to slaves. See [Section 12.4.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#).

13.5.7. Adding and Removing InnoDB Data and Log Files

This section describes what you can do when your InnoDB tablespace runs out of room or when you want to change the size of the log files.

The easiest way to increase the size of the InnoDB tablespace is to configure it from the beginning to be auto-extending. Specify the `autoextend` attribute for the last data file in the tablespace definition. Then InnoDB increases the size of that file automatically in 8MB increments when it runs out of space. The increment size can be changed by setting the value of the `innodb_autoextend_increment` system variable, which is measured in MB.

Alternatively, you can increase the size of your tablespace by adding another data file. To do this, you have to shut down the MySQL server, change the tablespace configuration to add a new data file to the end of `innodb_data_file_path`, and start the server again.

If your last data file was defined with the keyword `autoextend`, the procedure for reconfiguring the tablespace must take into account the size to which the last data file has grown. Obtain the size of the data file, round it down to the closest multiple of 1024×1024 bytes (= 1MB), and specify the rounded size explicitly in `innodb_data_file_path`. Then you can add another data file. Remember that only the last data file in the `innodb_data_file_path` can be specified as auto-extending.

As an example, assume that the tablespace has just one auto-extending data file `ibdata1`:

```
innodb_data_home_dir =  
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

Suppose that this data file, over time, has grown to 988MB. Here is the configuration line after modifying the original data file to not be auto-extending and adding another auto-extending data file:

```
innodb_data_home_dir =  
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

When you add a new file to the tablespace configuration, make sure that it does not exist. **InnoDB** will create and initialize the file when you restart the server.

Currently, you cannot remove a data file from the tablespace. To decrease the size of your tablespace, use this procedure:

1. Use `mysqldump` to dump all your **InnoDB** tables.
2. Stop the server.
3. Remove all the existing tablespace files.
4. Configure a new tablespace.
5. Restart the server.
6. Import the dump files.

If you want to change the number or the size of your **InnoDB** log files, use the following instructions. The procedure to use depends on the value of `innodb_fast_shutdown`:

- If `innodb_fast_shutdown` is not set to 2: You must stop the MySQL server and make sure that it shuts down without errors (to ensure that there is no information for outstanding transactions in the logs). Then copy the old log files into a safe place just in case something went wrong in the shutdown and you need them to recover the tablespace. Delete the old log files from the log file directory, edit `my.cnf` to change the log file configuration, and start the MySQL server again. `mysqld` sees that no log files exist at startup and tells you that it is creating new ones.
- If `innodb_fast_shutdown` is set to 2: You should shut down the server, set `innodb_fast_shutdown` to 1, and restart the server. The server should be allowed to recover. Then you should shut down the server again and follow the procedure described in the preceding item to change **InnoDB** log file size. Set `innodb_fast_shutdown` back to 2 and restart the server.

13.5.8. Backing Up and Recovering an **InnoDB** Database

The key to safe database management is making regular backups.

InnoDB Hot Backup is an online backup tool you can use to backup your **InnoDB** database while it is running. **InnoDB Hot Backup** does not require you to shut down your database and it does not set any locks or disturb your normal database processing. **InnoDB Hot Backup** is a non-free (commercial) add-on tool with an annual license fee of €390 per computer on which the MySQL server is run. See the [InnoDB Hot Backup home page](#) for detailed information and screenshots.

If you are able to shut down your MySQL server, you can make a binary backup that consists of all files used by **InnoDB** to manage its tables. Use the following procedure:

1. Shut down your MySQL server and make sure that it shuts down without errors.

2. Copy all your data files (`ibdata` files and `.ibd` files) into a safe place.
3. Copy all your `ib_logfile` files to a safe place.
4. Copy your `my.cnf` configuration file or files to a safe place.
5. Copy all the `.frm` files for your `InnoDB` tables to a safe place.

Replication works with `InnoDB` tables, so you can use MySQL replication capabilities to keep a copy of your database at database sites requiring high availability.

In addition to making binary backups as just described, you should also regularly make dumps of your tables with `mysqldump`. The reason for this is that a binary file might be corrupted without you noticing it. Dumped tables are stored into text files that are human-readable, so spotting table corruption becomes easier. Also, because the format is simpler, the chance for serious data corruption is smaller. `mysqldump` also has a `--single-transaction` option that you can use to make a consistent snapshot without locking out other clients.

To be able to recover your `InnoDB` database to the present from the binary backup just described, you have to run your MySQL server with binary logging turned on. Then you can apply the binary log to the backup database to achieve point-in-time recovery:

```
mysqlbinlog yourhostname-bin.123 | mysql
```

To recover from a crash of your MySQL server, the only requirement is to restart it. `InnoDB` automatically checks the logs and performs a roll-forward of the database to the present. `InnoDB` automatically rolls back uncommitted transactions that were present at the time of the crash. During recovery, `mysqld` displays output something like this:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

If your database gets corrupted or your disk fails, you have to do the recovery from a backup. In the case of corruption, you should first find a backup that is not corrupted. After restoring the base backup, do the recovery from the binary log files using `mysqlbinlog` and `mysql` to restore the changes performed after the backup was made.

In some cases of database corruption it is enough just to dump, drop, and re-create one or a few corrupt tables. You can use the `CHECK TABLE` SQL statement to check whether a table is corrupt, although `CHECK TABLE` naturally cannot detect every possible kind of corruption. You can use `innodb_tablespace_monitor` to check the integrity of the file space management inside the tablespace files.

In some cases, apparent database page corruption is actually due to the operating system corrupting its own file cache, and the data on disk may be okay. It is best first to try restarting your computer. Doing so may eliminate errors that appeared to be database page corruption.

13.5.8.1. Forcing `InnoDB` Recovery

If there is database page corruption, you may want to dump your tables from the database with `SELECT INTO OUTFILE`. Usually, most of the data obtained in this way is intact. Even so, the corruption may cause `SELECT * FROM tbl_name` statements or `InnoDB` background operations to crash or assert, or even to cause `InnoDB` roll-forward recovery to crash. However, you can force the `InnoDB` storage engine to start up while preventing background operations from running, so that you are able to dump your tables. For example, you can add the following line to the `[mysqld]` section of your option file before restarting the server:


```
[mysqld]  
innodb_force_recovery = 4
```

The allowable non-zero values for `innodb_force_recovery` follow. A larger number includes all precautions of smaller numbers. If you are able to dump your tables with an option value of at most 4, then you are relatively safe that only some data on corrupt individual pages is lost. A value of 6 is more drastic because database pages are left in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures.

- **1 (SRV_FORCE_IGNORE_CORRUPT)**

Let the server run even if it detects a corrupt page. Try to make `SELECT * FROM tbl_name` jump over corrupt index records and pages, which helps in dumping tables.

- **2 (SRV_FORCE_NO_BACKGROUND)**

Prevent the main thread from running. If a crash would occur during the purge operation, this recovery value prevents it.

- **3 (SRV_FORCE_NO_TRX_UNDO)**

Do not run transaction rollbacks after recovery.

- **4 (SRV_FORCE_NO_IBUF_MERGE)**

Prevent also insert buffer merge operations. If they would cause a crash, do not do them. Do not calculate table statistics.

- **5 (SRV_FORCE_NO_UNDO_LOG_SCAN)**

Do not look at undo logs when starting the database: `InnoDB` treats even incomplete transactions as committed.

- **6 (SRV_FORCE_NO_LOG_REDO)**

Do not do the log roll-forward in connection with recovery.

You can `SELECT` from tables to dump them, or `DROP` or `CREATE` tables even if forced recovery is used. If you know that a given table is causing a crash on rollback, you can drop it. You can also use this to stop a runaway rollback caused by a failing mass import or `ALTER TABLE`. You can kill the `mysqld` process and set `innodb_force_recovery` to 3 to bring the database up without the rollback, then `DROP` the table that is causing the runaway rollback.

The database must not otherwise be used with any non-zero value of `innodb_force_recovery`. As a safety measure, `InnoDB` prevents users from performing `INSERT`, `UPDATE`, or `DELETE` operations when `innodb_force_recovery` is greater than 0.

13.5.8.2. Checkpoints

`InnoDB` implements a checkpoint mechanism known as “fuzzy” checkpointing. `InnoDB` flushes modified database pages from the buffer pool in small batches. There is no need to flush the buffer pool in one single batch, which would in practice stop processing of user SQL statements during the checkpointing process.

During crash recovery, `InnoDB` looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are present in the disk image of the database. Then `InnoDB` scans the log files forward from the checkpoint, applying the logged modifications to the database.

`InnoDB` writes to its log files on a rotating basis. All committed modifications that make the database pages in the buffer pool different from the images on disk must be available in the log files in case `InnoDB` has to do a recovery. This means that when `InnoDB` starts to reuse a log file, it has to make sure that the database page images on disk contain the modifications logged in the log file that `InnoDB` is going to reuse. In other words, `InnoDB` must create a checkpoint and this often involves flushing of modified database pages to disk.

The preceding description explains why making your log files very large may save disk I/O in checkpointing. It often makes sense to set the total size of the log files as big as the buffer pool or even bigger. The drawback of using large log files is that crash recovery can take longer because there is more logged information to apply to the database.

13.5.9. Moving an `InnoDB` Database to Another Machine

On Windows, `InnoDB` always stores database and table names internally in lowercase. To move databases in a binary format from

Unix to Windows or from Windows to Unix, you should have all table and database names in lowercase. A convenient way to accomplish this is to add the following line to the `[mysqld]` section of your `my.cnf` or `my.ini` file before creating any databases or tables:

```
[mysqld]
lower_case_table_names=1
```

Like **MyISAM** data files, **InnoDB** data and log files are binary-compatible on all platforms having the same floating-point number format. You can move an **InnoDB** database simply by copying all the relevant files listed in [Section 13.5.8, “Backing Up and Recovering an InnoDB Database”](#). If the floating-point formats differ but you have not used **FLOAT** or **DOUBLE** data types in your tables, then the procedure is the same: simply copy the relevant files. If the formats differ and your tables contain floating-point data, you must use **mysqldump** to dump your tables on one machine and then import the dump files on the other machine.

One way to increase performance is to switch off autocommit mode when importing data, assuming that the tablespace has enough space for the big rollback segment that the import transactions generate. Do the commit only after importing a whole table or a segment of a table.

13.5.10. InnoDB Transaction Model and Locking

In the **InnoDB** transaction model, the goal is to combine the best properties of a multi-versioning database with traditional two-phase locking. **InnoDB** does locking on the row level and runs queries as non-locking consistent reads by default, in the style of Oracle. The lock table in **InnoDB** is stored so space-efficiently that lock escalation is not needed: Typically several users are allowed to lock every row in the database, or any random subset of the rows, without **InnoDB** running out of memory.

13.5.10.1. InnoDB Lock Modes

InnoDB implements standard row-level locking where there are two types of locks:

- A shared (**S**) lock allows a transaction to read a row (tuple).
- An exclusive (**X**) lock allows a transaction to update or delete a row.

If transaction **T1** holds a shared (**S**) lock on tuple **t**, then

- A request from some distinct transaction **T2** for an **S** lock on **t** can be granted immediately. As a result, both **T1** and **T2** hold an **S** lock on **t**.
- A request from some distinct transaction **T2** for an **X** lock on **t** cannot be granted immediately.

If a transaction **T1** holds an exclusive (**X**) lock on tuple **t**, then a request from some distinct transaction **T2** for a lock of either type on **t** cannot be granted immediately. Instead, transaction **T2** has to wait for transaction **T1** to release its lock on tuple **t**.

Additionally, **InnoDB** supports *multiple granularity locking* which allows coexistence of record locks and locks on entire tables. To make locking at multiple granularity levels practical, additional types of locks called *intention locks* are used. Intention locks are table locks in **InnoDB**. The idea behind intention locks is for a transaction to indicate which type of lock (shared or exclusive) it will require later for a row in that table. There are two types of intention locks used in **InnoDB** (assume that transaction **T** has requested a lock of the indicated type on table **R**):

- Intention shared (**IS**): Transaction **T** intends to set **S** locks on individual rows in table **R**.
- Intention exclusive (**IX**): Transaction **T** intends to set **X** locks on those rows.

The intention locking protocol is as follows:

- Before a given transaction can acquire an **S** lock on a given row, it must first acquire an **IS** or stronger lock on the table containing that row.
- Before a given transaction can acquire an **X** lock on a given row, it must first acquire an **IX** lock on the table containing that row.

These rules can be conveniently summarized by means of a *lock type compatibility matrix*:

	<i>X</i>	<i>IX</i>	<i>S</i>	<i>IS</i>
<i>X</i>	Conflict	Conflict	Conflict	Conflict
<i>IX</i>	Conflict	Compatible	Conflict	Compatible
<i>S</i>	Conflict	Conflict	Compatible	Compatible
<i>IS</i>	Conflict	Compatible	Compatible	Compatible

A lock is granted to a requesting transaction if it is compatible with existing locks. A lock is not granted to a requesting transaction if it conflicts with existing locks. A transaction waits until the conflicting existing lock is released. If a lock request conflicts with an existing lock and cannot be granted because it would cause deadlock, an error occurs.

Thus, intention locks do not block anything except full table requests (for example, `LOCK TABLES ... WRITE`). The main purpose of *IX* and *IS* locks is to show that someone is locking a row, or going to lock a row in the table.

The following example illustrates how an error can occur when a lock request would cause a deadlock. The example involves two clients, A and B.

First, client A creates a table containing one row, and then begins a transaction. Within the transaction, A obtains an *S* lock on the row by selecting it in share mode:

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);
Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE i = 1 LOCK IN SHARE MODE;
+-----+
| i     |
+-----+
|      1 |
+-----+
1 row in set (0.10 sec)
```

Next, client B begins a transaction and attempts to delete the row from the table:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM t WHERE i = 1;
```

The delete operation requires an *X* lock. The lock cannot be granted because it is incompatible with the *S* lock that client A holds, so the request goes on the queue of lock requests for the row and client B blocks.

Finally, client A also attempts to delete the row from the table:

```
mysql> DELETE FROM t WHERE i = 1;
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

Deadlock occurs here because client A needs an *X* lock to delete the row. However, that lock request cannot be granted because client B already has a request for an *X* lock and is waiting for client A to release its *S* lock. Nor can the *S* lock held by A be upgraded to an *X* lock because of the prior request by B for an *X* lock. As a result, *InnoDB* generates an error for client A and releases its locks. At that point, the lock request for client B can be granted and B deletes the row from the table.

13.5.10.2. *InnoDB* and AUTOCOMMIT

In *InnoDB*, all user activity occurs inside a transaction. If the autocommit mode is enabled, each SQL statement forms a single transaction on its own. By default, MySQL starts new connections with autocommit enabled.

If the autocommit mode is switched off with `SET AUTOCOMMIT = 0`, then we can consider that a user always has a transaction open. An SQL `COMMIT` or `ROLLBACK` statement ends the current transaction and a new one starts. A `COMMIT` means that the changes made

in the current transaction are made permanent and become visible to other users. A **ROLLBACK** statement, on the other hand, cancels all modifications made by the current transaction. Both statements release all **InnoDB** locks that were set during the current transaction.

If the connection has autocommit enabled, the user can still perform a multiple-statement transaction by starting it with an explicit **START TRANSACTION** or **BEGIN** statement and ending it with **COMMIT** or **ROLLBACK**.

13.5.10.3. InnoDB and TRANSACTION ISOLATION LEVEL

In terms of the SQL:1992 transaction isolation levels, the **InnoDB** default is **REPEATABLE READ**. **InnoDB** offers all four transaction isolation levels described by the SQL standard. You can set the default isolation level for all connections by using the **-transaction-isolation** option on the command line or in an option file. For example, you can set the option in the **[mysqld]** section of an option file like this:

```
[mysqld]
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
                        | REPEATABLE-READ | SERIALIZABLE}
```

A user can change the isolation level for a single session or for all new incoming connections with the **SET TRANSACTION** statement. Its syntax is as follows:

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
   {READ UNCOMMITTED | READ COMMITTED
    | REPEATABLE READ | SERIALIZABLE}
```

Note that there are hyphens in the level names for the **--transaction-isolation** option, but not for the **SET TRANSACTION** statement.

The default behavior is to set the isolation level for the next (not started) transaction. If you use the **GLOBAL** keyword, the statement sets the default transaction level globally for all new connections created from that point on (but not for existing connections). You need the **SUPER** privilege to do this. Using the **SESSION** keyword sets the default transaction level for all future transactions performed on the current connection.

Any client is free to change the session isolation level (even in the middle of a transaction), or the isolation level for the next transaction.

You can determine the global and session transaction isolation levels by checking the value of the **tx_isolation** system variable with these statements:

```
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;
```

In row-level locking, **InnoDB** uses next-key locking. That means that besides index records, **InnoDB** can also lock the “gap” preceding an index record to block insertions by other users immediately before the index record. A next-key lock refers to a lock that locks an index record and the gap before it. A gap lock refers to a lock that only locks a gap before some index record. Next-key locking for searches or index scans can be disabled by enabling the **innodb_locks_unsafe_for_binlog** system variable.

A detailed description of each isolation level in **InnoDB** follows:

- **READ UNCOMMITTED**

SELECT statements are performed in a non-locking fashion, but a possible earlier version of a record might be used. Thus, using this isolation level, such reads are not consistent. This is also called a “dirty read.” Otherwise, this isolation level works like **READ COMMITTED**.

- **READ COMMITTED**

A somewhat Oracle-like isolation level. All **SELECT ... FOR UPDATE** and **SELECT ... LOCK IN SHARE MODE** statements lock only the index records, not the gaps before them, and thus allow the free insertion of new records next to locked records. **UPDATE** and **DELETE** statements using a unique index with a unique search condition lock only the index record found, not the gap before it. In range-type **UPDATE** and **DELETE** statements, **InnoDB** must set next-key or gap locks and block insertions by other users to the gaps covered by the range. This is necessary because “phantom rows” must be blocked for MySQL replication and recovery to work.

Consistent reads behave as in Oracle: Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. See [Section 13.5.10.4, “Consistent Non-Locking Read”](#).

- **REPEATABLE READ**

This is the default isolation level of **InnoDB**. **SELECT ... FOR UPDATE**, **SELECT ... LOCK IN SHARE MODE**, **UPDATE**, and **DELETE** statements that use a unique index with a unique search condition lock only the index record found, not the gap before it. With other search conditions, these operations employ next-key locking, locking the index range scanned with next-key or gap locks, and block new insertions by other users.

In consistent reads, there is an important difference from the **READ COMMITTED** isolation level: All consistent reads within the same transaction read the same snapshot established by the first read. This convention means that if you issue several plain **SELECT** statements within the same transaction, these **SELECT** statements are consistent also with respect to each other. See [Section 13.5.10.4, “Consistent Non-Locking Read”](#).

- **SERIALIZABLE**

This level is like **REPEATABLE READ**, but **InnoDB** implicitly commits all plain **SELECT** statements to **SELECT ... LOCK IN SHARE MODE**.

In MySQL 5.1, if the **READ COMMITTED** isolation level is used or the **innodb_locks_unsafe_for_binlog** system variable is enabled, there is no **InnoDB** gap locking except in constraint checking. Also, record locks for non-matching rows are released after MySQL has evaluated the **WHERE** condition.

13.5.10.4. Consistent Non-Locking Read

A consistent read means that **InnoDB** uses multi-versioning to present to a query a snapshot of the database at a point in time. The query sees the changes made by those transactions that committed before that point of time, and no changes made by later or uncommitted transactions. The exception to this rule is that the query sees the changes made by earlier statements within the same transaction. Note that the exception to the rule causes the following anomaly: if you update some rows in a table, a **SELECT** will see the latest version of the updated rows, but it might also see older versions of any rows. If other users simultaneously update the same table, the anomaly means that you may see the table in a state that never existed in the database.

If you are running with the default **REPEATABLE READ** isolation level, all consistent reads within the same transaction read the snapshot established by the first such read in that transaction. You can get a fresher snapshot for your queries by committing the current transaction and after that issuing new queries.

Consistent read is the default mode in which **InnoDB** processes **SELECT** statements in **READ COMMITTED** and **REPEATABLE READ** isolation levels. A consistent read does not set any locks on the tables it accesses, and therefore other users are free to modify those tables at the same time a consistent read is being performed on the table.

Note that consistent read does not work over **DROP TABLE** and over **ALTER TABLE**. Consistent read does not work over **DROP TABLE** because MySQL can't use a table that has been dropped and **InnoDB** destroys the table. Consistent read does not work over **ALTER TABLE** because **ALTER TABLE** works by making a temporary copy of the original table and deleting the original table when the temporary copy is built. When you reissue a consistent read within a transaction, rows in the new table are not visible because those rows did not exist when the transaction's snapshot was taken.

InnoDB uses a consistent read for select in clauses like **INSERT INTO ... SELECT** and **UPDATE ... (SELECT)** that do not specify **FOR UPDATE** or **IN SHARE MODE** if the **innodb_locks_unsafe_for_binlog** option is set and the isolation level of the transaction is not set to serializable. Thus no locks are set to rows read from selected table. Otherwise, **InnoDB** uses stronger locks and the **SELECT** part acts like **READ COMMITTED**, where each consistent read, even within the same transaction, sets and reads its own fresh snapshot.

13.5.10.5. **SELECT ... FOR UPDATE** and **SELECT ... LOCK IN SHARE MODE** Locking Reads

In some circumstances, a consistent read is not convenient. For example, you might want to add a new row into your table **child**, and make sure that the child has a parent in table **parent**. The following example shows how to implement referential integrity in your application code.

Suppose that you use a consistent read to read the table **parent** and indeed see the parent of the child in the table. Can you safely add the child row to table **child**? No, because it may happen that meanwhile some other user deletes the parent row from the table **parent** without you being aware of it.

The solution is to perform the **SELECT** in a locking mode using **LOCK IN SHARE MODE**:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

Performing a read in share mode means that we read the latest available data, and set a shared mode lock on the rows we read. A shared mode lock prevents others from updating or deleting the row we have read. Also, if the latest data belongs to a yet uncommitted transaction of another client connection, we wait until that transaction commits. After we see that the preceding query returns the parent 'Jones', we can safely add the child record to the `child` table and commit our transaction.

Let us look at another example: We have an integer counter field in a table `child_codes` that we use to assign a unique identifier to each child added to table `child`. Obviously, using a consistent read or a shared mode read to read the present value of the counter is not a good idea because two users of the database may then see the same value for the counter, and a duplicate-key error occurs if two users attempt to add children with the same identifier to the table.

Here, `LOCK IN SHARE MODE` is not a good solution because if two users read the counter at the same time, at least one of them ends up in deadlock when attempting to update the counter.

In this case, there are two good ways to implement the reading and incrementing of the counter: (1) update the counter first by incrementing it by 1 and only after that read it, or (2) read the counter first with a lock mode `FOR UPDATE`, and increment after that. The latter approach can be implemented as follows:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

A `SELECT ... FOR UPDATE` reads the latest available data, setting exclusive locks on each row it reads. Thus, it sets the same locks a searched SQL `UPDATE` would set on the rows.

The preceding description is merely an example of how `SELECT ... FOR UPDATE` works. In MySQL, the specific task of generating a unique identifier actually can be accomplished using only a single access to the table:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

The `SELECT` statement merely retrieves the identifier information (specific to the current connection). It does not access any table.

Locks set by `IN SHARE MODE` and `FOR UPDATE` reads are released when the transaction is committed or rolled back.

Note

Locking of rows for update using `SELECT FOR UPDATE` only applies when `AUTOCOMMIT` is switched off. If `AUTO-COMMIT` is on, then the rows matching the specification are not locked.

13.5.10.6. Next-Key Locking: Avoiding the Phantom Problem

In row-level locking, `InnoDB` uses an algorithm called *next-key locking*. `InnoDB` performs the row-level locking in such a way that when it searches or scans an index of a table, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index record locks.

The next-key locks that `InnoDB` sets on index records also affect the “gap” before that index record. If a user has a shared or exclusive lock on record `R` in an index, another user cannot insert a new index record immediately before `R` in the index order. (A gap lock refers to a lock that only locks a gap before some index record.)

This next-key locking of gaps is done to prevent the so-called “phantom problem.” Suppose that you want to read and lock all children from the `child` table having an identifier value greater than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

Suppose that there is an index on the `id` column. The query scans that index starting from the first record where `id` is bigger than 100. If the locks set on the index records would not lock out inserts made in the gaps, a new row might meanwhile be inserted to the table. If you execute the same `SELECT` within the same transaction, you would see a new row in the result set returned by the query. This is contrary to the isolation principle of transactions: A transaction should be able to run so that the data it has read does not change during the transaction. If we regard a set of rows as a data item, the new “phantom” child would violate this isolation principle.

When `InnoDB` scans an index, it can also lock the gap after the last record in the index. Just that happens in the previous example: The locks set by `InnoDB` prevent any insert to the table where `id` would be bigger than 100.

You can use next-key locking to implement a uniqueness check in your application: If you read your data in share mode and do not see a duplicate for a row you are going to insert, then you can safely insert your row and know that the next-key lock set on the successor of

your row during the read prevents anyone meanwhile inserting a duplicate for your row. Thus, the next-key locking allows you to “lock” the non-existence of something in your table.

In MySQL 5.1, if the `READ COMMITTED` isolation level is used or the `innodb_locks_unsafe_for_binlog` system variable is enabled, there is no **InnoDB** gap locking except in constraint checking. Also, record locks for non-matching rows are released after MySQL has evaluated the `WHERE` condition.

13.5.10.7. An Example of Consistent Read in InnoDB

Suppose that you are running in the default `REPEATABLE READ` isolation level. When you issue a consistent read (that is, an ordinary `SELECT` statement), **InnoDB** gives your transaction a timepoint according to which your query sees the database. If another transaction deletes a row and commits after your timepoint was assigned, you do not see the row as having been deleted. Inserts and updates are treated similarly.

You can advance your timepoint by committing your transaction and then doing another `SELECT`.

This is called *multi-versioned concurrency control*.

	User A	User B
time	SET AUTOCOMMIT=0;	SET AUTOCOMMIT=0;
	SELECT * FROM t;	
	empty set	INSERT INTO t VALUES (1, 2);
v	SELECT * FROM t;	
	empty set	COMMIT;
	SELECT * FROM t;	
	empty set	
	COMMIT;	
	SELECT * FROM t;	

	1 2	

	1 row in set	

In this example, user A sees the row inserted by B only when B has committed the insert and A has committed as well, so that the timepoint is advanced past the commit of B.

If you want to see the “freshest” state of the database, you should use either the `READ COMMITTED` isolation level or a locking read:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

13.5.10.8. Locks Set by Different SQL Statements in InnoDB

A locking read, an `UPDATE`, or a `DELETE` generally set record locks on every index record that is scanned in the processing of the SQL statement. It does not matter if there are `WHERE` conditions in the statement that would exclude the row. **InnoDB** does not remember the exact `WHERE` condition, but only knows which index ranges were scanned. The record locks are normally next-key locks that also block inserts to the “gap” immediately before the record.

If the locks to be set are exclusive, **InnoDB** always retrieves also the clustered index record and sets a lock on it.

If you do not have indexes suitable for your statement and MySQL has to scan the whole table to process the statement, every row of the table becomes locked, which in turn blocks all inserts by other users to the table. It is important to create good indexes so that your queries do not unnecessarily need to scan many rows.

InnoDB sets specific types of locks as follows:

- `SELECT ... FROM` is a consistent read, reading a snapshot of the database and setting no locks unless the transaction isolation level is set to `SERIALIZABLE`. For `SERIALIZABLE` level, this sets shared next-key locks on the index records it encounters.
- `SELECT ... FROM ... LOCK IN SHARE MODE` sets shared next-key locks on all index records the read encounters.
- `SELECT ... FROM ... FOR UPDATE` sets exclusive next-key locks on all index records the read encounters and also on the

corresponding clustered index records if a secondary index is used in the search.

- **INSERT INTO ... VALUES (...)** sets an exclusive lock on the inserted row. Note that this lock is not a next-key lock and does not prevent other users from inserting to the gap before the inserted row. If a duplicate-key error occurs, a shared lock on the duplicate index record is set.
- While initializing a previously specified **AUTO_INCREMENT** column on a table, **InnoDB** sets an exclusive lock on the end of the index associated with the **AUTO_INCREMENT** column. In accessing the auto-increment counter, **InnoDB** uses a specific table lock mode **AUTO-INC** where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Note that other clients cannot insert into the table while the **AUTO-INC** table lock is held; see [Section 13.5.10.2, “InnoDB and AUTO-COMMIT”](#).

InnoDB fetches the value of a previously initialized **AUTO_INCREMENT** column without setting any locks.

- **INSERT INTO T SELECT ... FROM S WHERE ...** sets an exclusive (non-next-key) lock on each row inserted into **T**. **InnoDB** sets shared next-key locks on **S**, unless **innodb_locks_unsafe_for_binlog** is enabled, in which case it does the search on **S** as a consistent read. **InnoDB** has to set locks in the latter case: In roll-forward recovery from a backup, every SQL statement has to be executed in exactly the same way it was done originally.
- **CREATE TABLE ... SELECT ...** performs the **SELECT** as a consistent read or with shared locks, as in the previous item.
- **REPLACE** is done like an insert if there is no collision on a unique key. Otherwise, an exclusive next-key lock is placed on the row that has to be updated.
- **UPDATE ... WHERE ...** sets an exclusive next-key lock on every record the search encounters.
- **DELETE FROM ... WHERE ...** sets an exclusive next-key lock on every record the search encounters.
- If a **FOREIGN KEY** constraint is defined on a table, any insert, update, or delete that requires the constraint condition to be checked sets shared record-level locks on the records that it looks at to check the constraint. **InnoDB** also sets these locks in the case where the constraint fails.
- **LOCK TABLES** sets table locks, but it is the higher MySQL layer above the **InnoDB** layer that sets these locks. **InnoDB** is aware of table locks if **innodb_table_locks=1** (the default) and **AUTOCOMMIT=0**, and the MySQL layer above **InnoDB** knows about row-level locks. Otherwise, **InnoDB**'s automatic deadlock detection cannot detect deadlocks where such table locks are involved. Also, because the higher MySQL layer does not know about row-level locks, it is possible to get a table lock on a table where another user currently has row-level locks. However, this does not endanger transaction integrity, as discussed in [Section 13.5.10.10, “Deadlock Detection and Rollback”](#). See also [Section 13.5.16, “Restrictions on InnoDB Tables”](#).
- In MySQL 5.1, if the **READ COMMITTED** isolation level is used or the **innodb_locks_unsafe_for_binlog** system variable is enabled, there is no **InnoDB** gap locking except in constraint checking. Also, record locks for non-matching rows are released after MySQL has evaluated the **WHERE** condition.

13.5.10.9. Implicit Transaction Commit and Rollback

By default, MySQL begins each client connection with autocommit mode enabled. When autocommit is enabled, MySQL does a commit after each SQL statement if that statement did not return an error. If an SQL statement returns an error, the commit or rollback behavior depends on the error. See [Section 13.5.15, “InnoDB Error Handling”](#).

If you have the autocommit mode off and close a connection without explicitly committing the final transaction, MySQL rolls back that transaction.

Each of the following statements (and any synonyms for them) implicitly end a transaction, as if you had done a **COMMIT** before executing the statement:

- **ALTER EVENT, ALTER FUNCTION, ALTER FUNCTION, ALTER PROCEDURE, ALTER TABLE, BEGIN, CREATE DATABASE, CREATE EVENT, CREATE FUNCTION, CREATE INDEX, CREATE PROCEDURE, CREATE TABLE, DROP DATABASE, DROP EVENT, DROP FUNCTION, DROP INDEX, DROP PROCEDURE, DROP TABLE, LOAD DATA INFILE LOCK TABLES, RENAME TABLE, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE TABLE, UNLOCK TABLES.**
- Beginning with MySQL 5.1.3, the **ALTER VIEW, CREATE TRIGGER, CREATE USER, CREATE VIEW, DROP TRIGGER, DROP USER, DROP VIEW, and RENAME USER** statements cause an implicit commit.

- **UNLOCK TABLES** commits a transaction only if any tables currently have been locked with **LOCK TABLES**. This does not occur for **UNLOCK TABLES** following **FLUSH TABLES WITH READ LOCK** because the latter statement does not acquire table-level locks.
- The **CREATE TABLE** statement in **InnoDB** is processed as a single transaction. This means that a **ROLLBACK** from the user does not undo **CREATE TABLE** statements the user made during that transaction.
- **CREATE TABLE** and **DROP TABLE** do not commit a transaction if the **TEMPORARY** keyword is used. (This does not apply to other operations on temporary tables such as **CREATE INDEX**, which do cause a commit.)
- In MySQL 5.1.11 and earlier, **LOAD DATA INFILE** caused an implicit commit for all storage engines. Beginning with MySQL 5.1.12, it causes an implicit commit only for tables using the **NDB** storage engine. For more information, see [Bug#11151](#).

Transactions cannot be nested. This is a consequence of the implicit **COMMIT** performed for any current transaction when you issue a **START TRANSACTION** statement or one of its synonyms.

Statements that cause implicit commit cannot be used in an XA transaction while the transaction is in an **ACTIVE** state.

The **BEGIN** statement differs from the use of the **BEGIN** keyword that starts a **BEGIN ... END** compound statement. The latter does not cause an implicit commit. See [Section 19.2.5, “BEGIN ... END Compound Statement Syntax”](#).

13.5.10.10. Deadlock Detection and Rollback

InnoDB automatically detects a deadlock of transactions and rolls back a transaction or transactions to break the deadlock. **InnoDB** tries to pick small transactions to roll back, where the size of a transaction is determined by the number of rows inserted, updated, or deleted.

InnoDB is aware of table locks if **innodb_table_locks=1** (the default) and **AUTOCOMMIT=0**, and the MySQL layer above it knows about row-level locks. Otherwise, **InnoDB** cannot detect deadlocks where a table lock set by a MySQL **LOCK TABLES** statement or a lock set by a storage engine other than **InnoDB** is involved. You must resolve these situations by setting the value of the **innodb_lock_wait_timeout** system variable.

When **InnoDB** performs a complete rollback of a transaction, all locks set by the transaction are released. However, if just a single SQL statement is rolled back as a result of an error, some of the locks set by the statement may be preserved. This happens because **InnoDB** stores row locks in a format such that it cannot know afterward which lock was set by which statement.

13.5.10.11. How to Cope with Deadlocks

Deadlocks are a classic problem in transactional databases, but they are not dangerous unless they are so frequent that you cannot run certain transactions at all. Normally, you must write your applications so that they are always prepared to re-issue a transaction if it gets rolled back because of a deadlock.

InnoDB uses automatic row-level locking. You can get deadlocks even in the case of transactions that just insert or delete a single row. That is because these operations are not really “atomic”; they automatically set locks on the (possibly several) index records of the row inserted or deleted.

You can cope with deadlocks and reduce the likelihood of their occurrence with the following techniques:

- Use **SHOW ENGINE INNODB STATUS** to determine the cause of the latest deadlock. That can help you to tune your application to avoid deadlocks.
- Always be prepared to re-issue a transaction if it fails due to deadlock. Deadlocks are not dangerous. Just try again.
- Commit your transactions often. Small transactions are less prone to collision.
- If you are using locking reads (**SELECT ... FOR UPDATE** or **... LOCK IN SHARE MODE**), try using a lower isolation level such as **READ COMMITTED**.
- Access your tables and rows in a fixed order. Then transactions form well-defined queues and do not deadlock.
- Add well-chosen indexes to your tables. Then your queries need to scan fewer index records and consequently set fewer locks. Use **EXPLAIN SELECT** to determine which indexes the MySQL server regards as the most appropriate for your queries.

- Use less locking. If you can afford to allow a `SELECT` to return data from an old snapshot, do not add the clause `FOR UPDATE` or `LOCK IN SHARE MODE` to it. Using the `READ COMMITTED` isolation level is good here, because each consistent read within the same transaction reads from its own fresh snapshot.
- If nothing else helps, serialize your transactions with table-level locks. The correct way to use `LOCK TABLES` with transactional tables, such as `InnoDB` tables, is to set `AUTOCOMMIT = 0` and not to call `UNLOCK TABLES` until after you commit the transaction explicitly. For example, if you need to write to table `t1` and read from table `t2`, you can do this:

```
SET AUTOCOMMIT=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

Table-level locks make your transactions queue nicely, and deadlocks are avoided.

- Another way to serialize transactions is to create an auxiliary “semaphore” table that contains just a single row. Have each transaction update that row before accessing other tables. In that way, all transactions happen in a serial fashion. Note that the `InnoDB` instant deadlock detection algorithm also works in this case, because the serializing lock is a row-level lock. With MySQL table-level locks, the timeout method must be used to resolve deadlocks.

13.5.11. InnoDB Performance Tuning Tips

- In `InnoDB`, having a long `PRIMARY KEY` wastes a lot of disk space because its value must be stored with every secondary index record. (See [Section 13.5.13, “InnoDB Table and Index Structures”](#).) Create an `AUTO_INCREMENT` column as the primary key if your primary key is long.
- If the Unix `top` tool or the Windows Task Manager shows that the CPU usage percentage with your workload is less than 70%, your workload is probably disk-bound. Maybe you are making too many transaction commits, or the buffer pool is too small. Making the buffer pool bigger can help, but do not set it equal to more than 80% of physical memory.
- Wrap several modifications into one transaction. `InnoDB` must flush the log to disk at each transaction commit if that transaction made modifications to the database. The rotation speed of a disk is typically at most 167 revolutions/second, which constrains the number of commits to the same 167th of a second if the disk does not “fool” the operating system.
- If you can afford the loss of some of the latest committed transactions if a crash occurs, you can set the `innodb_flush_log_at_trx_commit` parameter to 0. `InnoDB` tries to flush the log once per second anyway, although the flush is not guaranteed. You should also set the value of `innodb_support_xa` to 0 which will reduce the number of disk flushes due to synchronizing on disk data and the binary log.
- Make your log files big, even as big as the buffer pool. When `InnoDB` has written the log files full, it has to write the modified contents of the buffer pool to disk in a checkpoint. Small log files cause many unnecessary disk writes. The drawback of big log files is that the recovery time is longer.
- Make the log buffer quite large as well (on the order of 8MB).
- Use the `VARCHAR` data type instead of `CHAR` if you are storing variable-length strings or if the column may contain many `NULL` values. A `CHAR(N)` column always takes `N` characters to store data, even if the string is shorter or its value is `NULL`. Smaller tables fit better in the buffer pool and reduce disk I/O.

When using `row_format=compact` (the default `InnoDB` record format in MySQL 5.1) and variable-length character sets, such as `utf8` or `sjis`, `CHAR(N)` will occupy a variable amount of space, at least `N` bytes.

- In some versions of GNU/Linux and Unix, flushing files to disk with the Unix `fsync()` call (which `InnoDB` uses by default) and other similar methods is surprisingly slow. If you are dissatisfied with database write performance, you might try setting the `innodb_flush_method` parameter to `O_DSYNC`. Although `O_DSYNC` seems to be slower on most systems, yours might not be one of them.
- When using the `InnoDB` storage engine on Solaris 10 for x86_64 architecture (AMD Opteron), it is important to mount any filesystems used for storing `InnoDB`-related files using the `forcedirectio` option. (The default on Solaris 10/x86_64 is *not* to use this option.) Failure to use `forcedirectio` causes a serious degradation of `InnoDB`'s speed and performance on this platform.

When using the `InnoDB` storage engine with a large `innodb_buffer_pool_size` value on any release of Solaris 2.6 and up and any platform (sparc/x86/x64/amd64), a significant performance gain can be achieved by placing `InnoDB` data files and log files

on raw devices or on a separate direct I/O UFS filesystem (using mount option `forcedirectio`; see `mount_ufs(1M)`). Users of the Veritas filesystem VxFS should use the mount option `convosync=direct`.

Other MySQL data files, such as those for **MyISAM** tables, should not be placed on a direct I/O filesystem. Executables or libraries *must not* be placed on a direct I/O filesystem.

- When importing data into **InnoDB**, make sure that MySQL does not have autocommit mode enabled because that requires a log flush to disk for every insert. To disable autocommit during your import operation, surround it with `SET AUTOCOMMIT` and `COMMIT` statements:

```
SET AUTOCOMMIT=0;
... SQL import statements ...
COMMIT;
```

If you use the `mysqldump` option `--opt`, you get dump files that are fast to import into an **InnoDB** table, even without wrapping them with the `SET AUTOCOMMIT` and `COMMIT` statements.

- Beware of big rollbacks of mass inserts: **InnoDB** uses the insert buffer to save disk I/O in inserts, but no such mechanism is used in a corresponding rollback. A disk-bound rollback can take 30 times as long to perform as the corresponding insert. Killing the database process does not help because the rollback starts again on server startup. The only way to get rid of a runaway rollback is to increase the buffer pool so that the rollback becomes CPU-bound and runs fast, or to use a special procedure. See [Section 13.5.8.1, “Forcing InnoDB Recovery”](#).
- Beware also of other big disk-bound operations. Use `DROP TABLE` and `CREATE TABLE` to empty a table, not `DELETE FROM tbl_name`.
- Use the multiple-row `INSERT` syntax to reduce communication overhead between the client and the server if you need to insert many rows:

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

This tip is valid for inserts into any table, not just **InnoDB** tables.

- If you have **UNIQUE** constraints on secondary keys, you can speed up table imports by temporarily turning off the uniqueness checks during the import session:

```
SET UNIQUE_CHECKS=0;
... import operation ...
SET UNIQUE_CHECKS=1;
```

For big tables, this saves a lot of disk I/O because **InnoDB** can use its insert buffer to write secondary index records in a batch. Be certain that the data contains no duplicate keys. `UNIQUE_CHECKS` allows but does not require storage engines to ignore duplicate keys.

- If you have **FOREIGN KEY** constraints in your tables, you can speed up table imports by turning the foreign key checks off for the duration of the import session:

```
SET FOREIGN_KEY_CHECKS=0;
... import operation ...
SET FOREIGN_KEY_CHECKS=1;
```

For big tables, this can save a lot of disk I/O.

- If you often have recurring queries for tables that are not updated frequently, use the query cache:

```
[mysqld]
query_cache_type = ON
query_cache_size = 10M
```

- Unlike **MyISAM**, **InnoDB** does not store an index cardinality value in its tables. Instead, **InnoDB** computes a cardinality for a table the first time it accesses it after startup. With a large number of tables, this might take significant time. It is the initial table open operation that is important, so to “warm up” a table for later use, you might want to use it immediately after start up by issuing a statement such as `SELECT 1 FROM tbl_name LIMIT 1`.

For optimization recommendations geared to your specific circumstances subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

13.5.11.1. SHOW ENGINE INNODB STATUS and the InnoDB Monitors

InnoDB includes InnoDB Monitors that print information about the InnoDB internal state. You can use the `SHOW ENGINE INNODB STATUS` SQL statement at any time to fetch the output of the standard InnoDB Monitor to your SQL client. This information is useful in performance tuning. (If you are using the `mysql` interactive SQL client, the output is more readable if you replace the usual semicolon statement terminator with `\G`.) For a discussion of InnoDB lock modes, see [Section 13.5.10.1, “InnoDB Lock Modes”](#).

```
mysql> SHOW ENGINE INNODB STATUS\G
```

Another way to use InnoDB Monitors is to let them periodically write data to the standard output of the `mysqld` server. In this case, no output is sent to clients. When switched on, InnoDB Monitors print data about every 15 seconds. Server output usually is directed to the `.err` log in the MySQL data directory. This data is useful in performance tuning. On Windows, you must start the server from a command prompt in a console window with the `--console` option if you want to direct the output to the window rather than to the error log.

Monitor output includes the following types of information:

- Table and record locks held by each active transaction
- Lock waits of a transactions
- Semaphore waits of threads
- Pending file I/O requests
- Buffer pool statistics
- Purge and insert buffer merge activity of the main InnoDB thread

To cause the standard InnoDB Monitor to write to the standard output of `mysqld`, use the following SQL statement:

```
CREATE TABLE innodb_monitor (a INT) ENGINE=INNODB;
```

The monitor can be stopped by issuing the following statement:

```
DROP TABLE innodb_monitor;
```

The `CREATE TABLE` syntax is just a way to pass a command to the InnoDB engine through MySQL's SQL parser: The only things that matter are the table name `innodb_monitor` and that it be an InnoDB table. The structure of the table is not relevant at all for the InnoDB Monitor. If you shut down the server, the monitor does not restart automatically when you restart the server. You must drop the monitor table and issue a new `CREATE TABLE` statement to start the monitor. (This syntax may change in a future release.)

You can use `innodb_lock_monitor` in a similar fashion. This is the same as `innodb_monitor`, except that it also provides a great deal of lock information. A separate `innodb_tablespace_monitor` prints a list of created file segments existing in the tablespace and validates the tablespace allocation data structures. In addition, there is `innodb_table_monitor` with which you can print the contents of the InnoDB internal data dictionary.

A sample of InnoDB Monitor output:

```
mysql> SHOW ENGINE INNODB STATUS\G
***** 1. row *****
Status:
=====
030709 13:00:59 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 18 seconds
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 413452, signal count 378357
--Thread 32782 has waited at btr0sea.c line 1477 for 0.00 seconds the
semaphore: X-lock on RW-latch at 41a28668 created in file btr0sea.c line 135
a writer (thread id 32782) has reserved it in mode wait exclusive
number of readers 1, waiters flag 1
Last time read locked in file btr0sea.c line 731
```

[illegible]

```

----- TRX HAS BEEN WAITING 0 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 65556 n bits 232 table test/ibtest11a index
PRIMARY trx id 0 290328327 lock_mode X waiting
Record lock, heap no 1 RECORD: info bits 0 0: len 9; hex 73757072656d756d00;
asc supremum.;
-----
---TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread id
34831 rollback of SQL statement
ROLLING BACK 14 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk', 'khDk')
---TRANSACTION 0 290327208, ACTIVE 1 sec, process no 3190, OS thread id
32782
58 lock struct(s), heap size 5504, undo log entries 159
MySQL thread id 23, query id 4668732 localhost heikki update
REPLACE INTO alex1 VALUES(86, 46, 538, 'aa95666', 'bb', 'c95666', 'd9486t',
'e200498f', 'g86814', 'h538', date_format('2001-04-03 12:54:22', '%Y-%m-%d
%H:%i')),
---TRANSACTION 0 290323325, ACTIVE 3 sec, process no 3185, OS thread id
30733 inserting
4 lock struct(s), heap size 1024, undo log entries 165
MySQL thread id 21, query id 4668735 localhost heikki update
INSERT INTO alex1 VALUES(NULL, 49, NULL, 'aa42837', '', 'c56319', 'd1719t', '',
NULL, 'h321', NULL, NULL, 7.31, 7.31, 7.31, 200)
-----
FILE I/O
-----
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
  ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
151671 OS file reads, 94747 OS file writes, 8750 OS fsyncs
25.44 reads/s, 18494 avg bytes/read, 17.55 writes/s, 2.33 fsyncs/s
-----
INSERT BUFFER AND ADAPTIVE HASH INDEX
-----
Ibuf for space 0: size 1, free list len 19, seg size 21,
85004 inserts, 85004 merged recs, 26669 merges
Hash table size 207619, used cells 14461, node heap has 16 buffer(s)
1877.67 hash searches/s, 5121.10 non-hash searches/s
---
LOG
---
Log sequence number 18 1212842764
Log flushed up to 18 1212665295
Last checkpoint at 18 1135877290
0 pending log writes, 0 pending chkp writes
4341 log i/o's done, 1.22 log i/o's/second
-----
BUFFER POOL AND MEMORY
-----
Total memory allocated 84966343; in additional pool allocated 1402624
Buffer pool size 3200
Free buffers 110
Database pages 3074
Modified db pages 2674
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 171380, created 51968, written 194688
28.72 reads/s, 20.72 creates/s, 47.55 writes/s
Buffer pool hit rate 999 / 1000
-----
ROW OPERATIONS
-----
0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging
Number of rows inserted 3738558, updated 127415, deleted 33707, read 755779
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
-----
END OF INNODB MONITOR OUTPUT
=====

```

Some notes on the output:

- If the **TRANSACTIONS** section reports lock waits, your applications may have lock contention. The output can also help to trace the reasons for transaction deadlocks.
- The **SEMAPHORES** section reports threads waiting for a semaphore and statistics on how many times threads have needed a spin or a wait on a mutex or a rw-lock semaphore. A large number of threads waiting for semaphores may be a result of disk I/O, or contention problems inside **InnoDB**. Contention can be due to heavy parallelism of queries or problems in operating system thread scheduling. Setting **innodb_thread_concurrency** smaller than the default value can help in such situations.

- The **BUFFER POOL AND MEMORY** section gives you statistics on pages read and written. You can calculate from these numbers how many data file I/O operations your queries currently are doing.
- The **ROW OPERATIONS** section shows what the main thread is doing.

InnoDB sends diagnostic output to `stderr` or to files rather than to `stdout` or fixed-size memory buffers, to avoid potential buffer overflows. As a side effect, the output of **SHOW ENGINE INNODB STATUS** is written to a status file in the MySQL data directory every fifteen seconds. The name of the file is `innodb_status.pid`, where `pid` is the server process ID. **InnoDB** removes the file for a normal shutdown. If abnormal shutdowns have occurred, instances of these status files may be present and must be removed manually. Before removing them, you might want to examine them to see whether they contain useful information about the cause of abnormal shutdowns. The `innodb_status.pid` file is created only if the configuration option `innodb_status_file=1` is set.

13.5.12. Implementation of Multi-Versioning

Because **InnoDB** is a multi-versioned storage engine, it must keep information about old versions of rows in the tablespace. This information is stored in a data structure called a *rollback segment* (after an analogous data structure in Oracle).

Internally, **InnoDB** adds two fields to each row stored in the database. A 6-byte field indicates the transaction identifier for the last transaction that inserted or updated the row. Also, a deletion is treated internally as an update where a special bit in the row is set to mark it as deleted. Each row also contains a 7-byte field called the roll pointer. The roll pointer points to an undo log record written to the rollback segment. If the row was updated, the undo log record contains the information necessary to rebuild the content of the row before it was updated.

InnoDB uses the information in the rollback segment to perform the undo operations needed in a transaction rollback. It also uses the information to build earlier versions of a row for a consistent read.

Undo logs in the rollback segment are divided into insert and update undo logs. Insert undo logs are needed only in transaction rollback and can be discarded as soon as the transaction commits. Update undo logs are used also in consistent reads, but they can be discarded only after there is no transaction present for which **InnoDB** has assigned a snapshot that in a consistent read could need the information in the update undo log to build an earlier version of a database row.

You must remember to commit your transactions regularly, including those transactions that issue only consistent reads. Otherwise, **InnoDB** cannot discard data from the update undo logs, and the rollback segment may grow too big, filling up your tablespace.

The physical size of an undo log record in the rollback segment is typically smaller than the corresponding inserted or updated row. You can use this information to calculate the space need for your rollback segment.

In the **InnoDB** multi-versioning scheme, a row is not physically removed from the database immediately when you delete it with an SQL statement. Only when **InnoDB** can discard the update undo log record written for the deletion can it also physically remove the corresponding row and its index records from the database. This removal operation is called a purge, and it is quite fast, usually taking the same order of time as the SQL statement that did the deletion.

In a scenario where the user inserts and deletes rows in smallish batches at about the same rate in the table, it is possible that the purge thread starts to lag behind, and the table grows bigger and bigger, making everything disk-bound and very slow. Even if the table carries just 10MB of useful data, it may grow to occupy 10GB with all the “dead” rows. In such a case, it would be good to throttle new row operations, and allocate more resources to the purge thread. The `innodb_max_purge_lag` system variable exists for exactly this purpose. See [Section 13.5.4, “InnoDB Startup Options and System Variables”](#), for more information.

13.5.13. InnoDB Table and Index Structures

MySQL stores its data dictionary information for tables in `.frm` files in database directories. This is true for all MySQL storage engines. But every **InnoDB** table also has its own entry in the **InnoDB** internal data dictionary inside the tablespace. When MySQL drops a table or a database, it has to delete both an `.frm` file or files, and the corresponding entries inside the **InnoDB** data dictionary. This is the reason why you cannot move **InnoDB** tables between databases simply by moving the `.frm` files.

Every **InnoDB** table has a special index called the *clustered index* where the data for the rows is stored. If you define a **PRIMARY KEY** on your table, the index of the primary key is the clustered index.

If you do not define a **PRIMARY KEY** for your table, MySQL picks the first **UNIQUE** index that has only **NOT NULL** columns as the primary key and **InnoDB** uses it as the clustered index. If there is no such index in the table, **InnoDB** internally generates a clustered index where the rows are ordered by the row ID that **InnoDB** assigns to the rows in such a table. The row ID is a 6-byte field that increases monotonically as new rows are inserted. Thus, the rows ordered by the row ID are physically in insertion order.

Accessing a row through the clustered index is fast because the row data is on the same page where the index search leads. If a table is large, the clustered index architecture often saves a disk I/O when compared to the traditional solution. (In many database systems, data storage uses a different page from the index record.)

In **InnoDB**, the records in non-clustered indexes (also called secondary indexes) contain the primary key value for the row. **InnoDB** uses this primary key value to search for the row from the clustered index. Note that if the primary key is long, the secondary indexes use more space.

InnoDB compares **CHAR** and **VARCHAR** strings of different lengths such that the remaining length in the shorter string is treated as if padded with spaces.

13.5.13.1. Physical Structure of an Index

All **InnoDB** indexes are B-trees where the index records are stored in the leaf pages of the tree. The default size of an index page is 16KB. When new records are inserted, **InnoDB** tries to leave 1/16 of the page free for future insertions and updates of the index records.

If index records are inserted in a sequential order (ascending or descending), the resulting index pages are about 15/16 full. If records are inserted in a random order, the pages are from 1/2 to 15/16 full. If the fill factor of an index page drops below 1/2, **InnoDB** tries to contract the index tree to free the page.

13.5.13.2. Insert Buffering

It is a common situation in database applications that the primary key is a unique identifier and new rows are inserted in the ascending order of the primary key. Thus, the insertions to the clustered index do not require random reads from a disk.

On the other hand, secondary indexes are usually non-unique, and insertions into secondary indexes happen in a relatively random order. This would cause a lot of random disk I/O operations without a special mechanism used in **InnoDB**.

If an index record should be inserted to a non-unique secondary index, **InnoDB** checks whether the secondary index page is in the buffer pool. If that is the case, **InnoDB** does the insertion directly to the index page. If the index page is not found in the buffer pool, **InnoDB** inserts the record to a special insert buffer structure. The insert buffer is kept so small that it fits entirely in the buffer pool, and insertions can be done very fast.

Periodically, the insert buffer is merged into the secondary index trees in the database. Often it is possible to merge several insertions to the same page of the index tree, saving disk I/O operations. It has been measured that the insert buffer can speed up insertions into a table up to 15 times.

The insert buffer merging may continue to happen *after* the inserting transaction has been committed. In fact, it may continue to happen after a server shutdown and restart (see [Section 13.5.8.1, “Forcing InnoDB Recovery”](#)).

The insert buffer merging may take many hours, when many secondary indexes must be updated, and many rows have been inserted. During this time, disk I/O will be increased, which can cause significant slowdown on disk-bound queries. Another significant background I/O operation is the purge thread (see [Section 13.5.12, “Implementation of Multi-Versioning”](#)).

13.5.13.3. Adaptive Hash Indexes

If a table fits almost entirely in main memory, the fastest way to perform queries on it is to use hash indexes. **InnoDB** has a mechanism that monitors index searches made to the indexes defined for a table. If **InnoDB** notices that queries could benefit from building a hash index, it does so automatically.

Note that the hash index is always built based on an existing B-tree index on the table. **InnoDB** can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches that **InnoDB** observes for the B-tree index. A hash index can be partial: It is not required that the whole B-tree index is cached in the buffer pool. **InnoDB** builds hash indexes on demand for those pages of the index that are often accessed.

In a sense, **InnoDB** tailors itself through the adaptive hash index mechanism to amply main memory, coming closer to the architecture of main-memory databases.

13.5.13.4. Physical Row Structure

The physical record structure for **InnoDB** tables is dependent on the row format specified when the table was created. For MySQL 5.1, by default **InnoDB** uses the **COMPACT** format, but the **REDUNDANT** format is available to retain compatibility with older versions of MySQL.

Records in InnoDB `ROW_FORMAT=REDUNDANT` tables have the following characteristics:

- Each index record contains a six-byte header. The header is used to link together consecutive records, and also in row-level locking.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a six-byte field for the transaction ID and a seven-byte field for the roll pointer.
- If no primary key was defined for a table, each clustered index record also contains a six-byte row ID field.
- Each secondary index record contains also all the fields defined for the clustered index key.
- A record contains also a pointer to each field of the record. If the total length of the fields in a record is less than 128 bytes, the pointer is one byte; otherwise, two bytes. The array of these pointers is called the record directory. The area where these pointers point is called the data part of the record.
- Internally, InnoDB stores fixed-length character columns such as `CHAR(10)` in a fixed-length format. InnoDB truncates trailing spaces from `VARCHAR` columns.
- An SQL `NULL` value reserves 1 or 2 bytes in the record directory. Besides that, an SQL `NULL` value reserves zero bytes in the data part of the record if stored in a variable length column. In a fixed-length column, it reserves the fixed length of the column in the data part of the record. The motivation behind reserving the fixed space for `NULL` values is that it enables an update of the column from `NULL` to a non-`NULL` value to be done in place without causing fragmentation of the index page.

Records in InnoDB `ROW_FORMAT=COMPACT` tables have the following characteristics:

- Each index record contains a five-byte header that may be preceded by a variable-length header. The header is used to link together consecutive records, and also in row-level locking.
- The record header contains a bit vector for indicating `NULL` columns. The bit vector occupies $(n\_nullable+7)/8$ bytes. Columns that are `NULL` will not occupy other space than the bit in this vector.
- For each non-`NULL` variable-length field, the record header contains the length of the column in one or two bytes. Two bytes will only be needed if part of the column is stored externally or the maximum length exceeds 255 bytes and the actual length exceeds 127 bytes.
- The record header is followed by the data contents of the columns. Columns that are `NULL` are omitted.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a six-byte field for the transaction ID and a seven-byte field for the roll pointer.
- If no primary key was defined for a table, each clustered index record also contains a six-byte row ID field.
- Each secondary index record contains also all the fields defined for the clustered index key.
- Internally, InnoDB stores fixed-length, fixed-width character columns such as `CHAR(10)` in a fixed-length format. InnoDB truncates trailing spaces from `VARCHAR` columns.
- Internally, InnoDB attempts to store UTF-8 `CHAR(n)` columns in `n` bytes by trimming trailing spaces. In `ROW_FORMAT=REDUNDANT`, such columns occupy $3*n$ bytes. The motivation behind reserving the minimum space `n` is that it in many cases enables an update of the column to be done in place without causing fragmentation of the index page.

The presence of the compact row format decreases row storage space by about 20% at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed it is likely to be faster. If it is a rare case that is limited by CPU speed, it might be slower.

13.5.14. InnoDB File Space Management and Disk I/O

13.5.14.1. InnoDB Disk I/O

InnoDB uses simulated asynchronous disk I/O: InnoDB creates a number of threads to take care of I/O operations, such as read-ahead.

There are two read-ahead heuristics in **InnoDB**:

- In sequential read-ahead, if **InnoDB** notices that the access pattern to a segment in the tablespace is sequential, it posts in advance a batch of reads of database pages to the I/O system.
- In random read-ahead, if **InnoDB** notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool, it posts the remaining reads to the I/O system.

InnoDB uses a novel file flush technique called *doublewrite*. It adds safety to recovery following an operating system crash or a power outage, and improves performance on most varieties of Unix by reducing the need for `fsync()` operations.

Doublewrite means that before writing pages to a data file, **InnoDB** first writes them to a contiguous tablespace area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed does **InnoDB** write the pages to their proper positions in the data file. If the operating system crashes in the middle of a page write, **InnoDB** can later find a good copy of the page from the doublewrite buffer during recovery.

13.5.14.2. File Space Management

The data files that you define in the configuration file form the tablespace of **InnoDB**. The files are simply concatenated to form the tablespace. There is no striping in use. Currently, you cannot define where within the tablespace your tables are allocated. However, in a newly created tablespace, **InnoDB** allocates space starting from the first data file.

The tablespace consists of database pages with a default size of 16KB. The pages are grouped into extents of 64 consecutive pages. The “files” inside a tablespace are called *segments* in **InnoDB**. The term “rollback segment” is somewhat confusing because it actually contains many tablespace segments.

Two segments are allocated for each index in **InnoDB**. One is for non-leaf nodes of the B-tree, the other is for the leaf nodes. The idea here is to achieve better sequentiality for the leaf nodes, which contain the data.

When a segment grows inside the tablespace, **InnoDB** allocates the first 32 pages to it individually. After that **InnoDB** starts to allocate whole extents to the segment. **InnoDB** can add to a large segment up to 4 extents at a time to ensure good sequentiality of data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an **InnoDB** tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you ask for available free space in the tablespace by issuing a `SHOW TABLE STATUS` statement, **InnoDB** reports the extents that are definitely free in the tablespace. **InnoDB** always reserves some extents for cleanup and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, **InnoDB** contracts the corresponding B-tree indexes. Whether the freed space becomes available for other users depends on whether the pattern of deletes frees individual pages or extents to the tablespace. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows are physically removed only in an (automatic) purge operation after they are no longer needed for transaction rollbacks or consistent reads. (See [Section 13.5.12, “Implementation of Multi-Versioning”](#).)

13.5.14.3. Defragmenting a Table

If there are random insertions into or deletions from the indexes of a table, the indexes may become fragmented. Fragmentation means that the physical ordering of the index pages on the disk is not close to the index ordering of the records on the pages, or that there are many unused pages in the 64-page blocks that were allocated to the index.

A symptom of fragmentation is that a table takes more space than it “should” take. How much that is exactly, is difficult to determine. All **InnoDB** data and indexes are stored in B-trees, and their fill factor may vary from 50% to 100%. Another symptom of fragmentation is that a table scan such as this takes more time than it “should” take:

```
SELECT COUNT(*) FROM t WHERE a_non_indexed_column <> 12345;
```

(In the preceding query, we are “fooling” the SQL optimizer into scanning the clustered index, rather than a secondary index.) Most disks can read 10 to 50MB/s, which can be used to estimate how fast a table scan should run.

It can speed up index scans if you periodically perform a “null” `ALTER TABLE` operation:

```
ALTER TABLE tbl_name ENGINE=INNODB
```

That causes MySQL to rebuild the table. Another way to perform a defragmentation operation is to use `mysqldump` to dump the table to a text file, drop the table, and reload it from the dump file.

If the insertions to an index are always ascending and records are deleted only from the end, the **InnoDB** filespace management algorithm guarantees that fragmentation in the index does not occur.

13.5.15. InnoDB Error Handling

Error handling in **InnoDB** is not always the same as specified in the SQL standard. According to the standard, any error during an SQL statement should cause the rollback of that statement. **InnoDB** sometimes rolls back only part of the statement, or the whole transaction. The following items describe how **InnoDB** performs error handling:

- If you run out of file space in the tablespace, a MySQL `Table is full` error occurs and **InnoDB** rolls back the SQL statement.
- A transaction deadlock causes **InnoDB** to roll back the entire transaction. In the case of a lock wait timeout, **InnoDB** rolls back only the most recent SQL statement.

When a transaction rollback occurs due to a deadlock or lock wait timeout, it cancels the effect of the statements within the transaction. But if the start-transaction statement was `START TRANSACTION` or `BEGIN` statement, rollback does not cancel that statement. Further SQL statements become part of the transaction until the occurrence of `COMMIT`, `ROLLBACK`, or some SQL statement that causes an implicit commit.

- A duplicate-key error rolls back the SQL statement, if you have not specified the `IGNORE` option in your statement.
- A `row too long error` rolls back the SQL statement.
- Other errors are mostly detected by the MySQL layer of code (above the **InnoDB** storage engine level), and they roll back the corresponding SQL statement. Locks are not released in a rollback of a single SQL statement.

During implicit rollbacks, as well as during the execution of an explicit `ROLLBACK` SQL command, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the relevant connection.

13.5.15.1. InnoDB Error Codes

The following is a non-exhaustive list of common **InnoDB**-specific errors that you may encounter, with information about why each occurs and how to resolve the problem.

- **1005 (ER_CANT_CREATE_TABLE)**
Cannot create table. If the error message refers to `errno` 150, table creation failed because a foreign key constraint was not correctly formed. If the error message refers to `errno` -1, table creation probably failed because the table includes a column name that matched the name of an internal **InnoDB** table.
- **1016 (ER_CANT_OPEN_FILE)**
Cannot find the **InnoDB** table from the **InnoDB** data files, although the `.frm` file for the table exists. See [Section 13.5.17.1, “Troubleshooting InnoDB Data Dictionary Operations”](#).
- **1114 (ER_RECORD_FILE_FULL)**
InnoDB has run out of free space in the tablespace. You should reconfigure the tablespace to add a new data file.
- **1205 (ER_LOCK_WAIT_TIMEOUT)**
Lock wait timeout expired. Transaction was rolled back.
- **1213 (ER_LOCK_DEADLOCK)**
Transaction deadlock. You should rerun the transaction.
- **1216 (ER_NO_REFERENCED_ROW)**

You are trying to add a row but there is no parent row, and a foreign key constraint fails. You should add the parent row first.

- [1217 \(ER_ROW_IS_REFERENCED\)](#)

You are trying to delete a parent row that has children, and a foreign key constraint fails. You should delete the children first.

13.5.15.2. Operating System Error Codes

To print the meaning of an operating system error number, use the [perror](#) program that comes with the MySQL distribution.

The following table provides a list of some common Linux system error codes. For a more complete list, see [Linux source code](#).

- [1 \(EPERM\)](#)
Operation not permitted
- [2 \(ENOENT\)](#)
No such file or directory
- [3 \(ESRCH\)](#)
No such process
- [4 \(EINTR\)](#)
Interrupted system call
- [5 \(EIO\)](#)
I/O error
- [6 \(ENXIO\)](#)
No such device or address
- [7 \(E2BIG\)](#)
Arg list too long
- [8 \(ENOEXEC\)](#)
Exec format error
- [9 \(EBADF\)](#)
Bad file number
- [10 \(ECHILD\)](#)
No child processes
- [11 \(EAGAIN\)](#)
Try again
- [12 \(ENOMEM\)](#)
Out of memory
- [13 \(EACCES\)](#)
Permission denied

- [14 \(EFAULT\)](#)
Bad address
- [15 \(ENOTBLK\)](#)
Block device required
- [16 \(EBUSY\)](#)
Device or resource busy
- [17 \(EEXIST\)](#)
File exists
- [18 \(EXDEV\)](#)
Cross-device link
- [19 \(ENODEV\)](#)
No such device
- [20 \(ENOTDIR\)](#)
Not a directory
- [21 \(EISDIR\)](#)
Is a directory
- [22 \(EINVAL\)](#)
Invalid argument
- [23 \(ENFILE\)](#)
File table overflow
- [24 \(EMFILE\)](#)
Too many open files
- [25 \(ENOTTY\)](#)
Inappropriate ioctl for device
- [26 \(ETXTBSY\)](#)
Text file busy
- [27 \(EFBIG\)](#)
File too large
- [28 \(ENOSPC\)](#)
No space left on device
- [29 \(ESPIPE\)](#)
Illegal seek
- [30 \(EROFS\)](#)
Read-only file system

- 31 (EMLINK)

Too many links

The following table provides a list of some common Windows system error codes. For a complete list see the [Microsoft Web site](#).

- 1 (ERROR_INVALID_FUNCTION)

Incorrect function.

- 2 (ERROR_FILE_NOT_FOUND)

The system cannot find the file specified.

- 3 (ERROR_PATH_NOT_FOUND)

The system cannot find the path specified.

- 4 (ERROR_TOO_MANY_OPEN_FILES)

The system cannot open the file.

- 5 (ERROR_ACCESS_DENIED)

Access is denied.

- 6 (ERROR_INVALID_HANDLE)

The handle is invalid.

- 7 (ERROR_ARENA_TRASHED)

The storage control blocks were destroyed.

- 8 (ERROR_NOT_ENOUGH_MEMORY)

Not enough storage is available to process this command.

- 9 (ERROR_INVALID_BLOCK)

The storage control block address is invalid.

- 10 (ERROR_BAD_ENVIRONMENT)

The environment is incorrect.

- 11 (ERROR_BAD_FORMAT)

An attempt was made to load a program with an incorrect format.

- 12 (ERROR_INVALID_ACCESS)

The access code is invalid.

- 13 (ERROR_INVALID_DATA)

The data is invalid.

- 14 (ERROR_OUTOFMEMORY)

Not enough storage is available to complete this operation.

- 15 (ERROR_INVALID_DRIVE)

The system cannot find the drive specified.

- **16 (ERROR_CURRENT_DIRECTORY)**
The directory cannot be removed.
- **17 (ERROR_NOT_SAME_DEVICE)**
The system cannot move the file to a different disk drive.
- **18 (ERROR_NO_MORE_FILES)**
There are no more files.
- **19 (ERROR_WRITE_PROTECT)**
The media is write protected.
- **20 (ERROR_BAD_UNIT)**
The system cannot find the device specified.
- **21 (ERROR_NOT_READY)**
The device is not ready.
- **22 (ERROR_BAD_COMMAND)**
The device does not recognize the command.
- **23 (ERROR_CRC)**
Data error (cyclic redundancy check).
- **24 (ERROR_BAD_LENGTH)**
The program issued a command but the command length is incorrect.
- **25 (ERROR_SEEK)**
The drive cannot locate a specific area or track on the disk.
- **26 (ERROR_NOT_DOS_DISK)**
The specified disk or diskette cannot be accessed.
- **27 (ERROR_SECTOR_NOT_FOUND)**
The drive cannot find the sector requested.
- **28 (ERROR_OUT_OF_PAPER)**
The printer is out of paper.
- **29 (ERROR_WRITE_FAULT)**
The system cannot write to the specified device.
- **30 (ERROR_READ_FAULT)**
The system cannot read from the specified device.
- **31 (ERROR_GEN_FAILURE)**
A device attached to the system is not functioning.
- **32 (ERROR_SHARING_VIOLATION)**
The process cannot access the file because it is being used by another process.

- **33 (ERROR_LOCK_VIOLATION)**

The process cannot access the file because another process has locked a portion of the file.

- **34 (ERROR_WRONG_DISK)**

The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.

- **36 (ERROR_SHARING_BUFFER_EXCEEDED)**

Too many files opened for sharing.

- **38 (ERROR_HANDLE_EOF)**

Reached the end of the file.

- **39 (ERROR_HANDLE_DISK_FULL)**

The disk is full.

- **87 (ERROR_INVALID_PARAMETER)**

The parameter is incorrect. (If this error occurs on Windows and you have enabled `innodb_file_per_table` in a server option file, add the line `innodb_flush_method=unbuffered` to the file as well.)

- **112 (ERROR_DISK_FULL)**

The disk is full.

- **123 (ERROR_INVALID_NAME)**

The filename, directory name, or volume label syntax is incorrect.

- **1450 (ERROR_NO_SYSTEM_RESOURCES)**

Insufficient system resources exist to complete the requested service.

13.5.16. Restrictions on InnoDB Tables

- **Warning:** Do not convert MySQL system tables in the `mysql` database from `MyISAM` to `InnoDB` tables! This is an unsupported operation. If you do this, MySQL does not restart until you restore the old system tables from a backup or re-generate them with the `mysql_install_db` script.

Warning

It is not a good idea to configure `InnoDB` to use datafiles or logfiles on NFS volumes. Otherwise, the files might be locked by other processes and become unavailable for use by MySQL.

- A table cannot contain more than 1000 columns.
- The internal maximum key length is 3500 bytes, but MySQL itself restricts this to 1024 bytes.
- The maximum row length, except for `VARBINARY`, `VARCHAR`, `BLOB` and `TEXT` columns, is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes. `LONGLOB` and `LONGTEXT` columns must be less than 4GB, and the total row length, including also `BLOB` and `TEXT` columns, must be less than 4GB. `InnoDB` stores the first 768 bytes of a `VARBINARY`, `VARCHAR`, `BLOB`, or `TEXT` column in the row, and the rest into separate pages.
- Although `InnoDB` supports row sizes larger than 65535 internally, you cannot define a row containing `VARBINARY` or `VARCHAR` columns with a combined size larger than 65535:

```
mysql> CREATE TABLE t (a VARCHAR(8000), b VARCHAR(10000),
-> c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
-> f VARCHAR(10000), g VARCHAR(10000)) ENGINE=InnoDB;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

- On some older operating systems, files must be less than 2GB. This is not a limitation of **InnoDB** itself, but if you require a large tablespace, you will need to configure it using several smaller data files rather than one or a file large data files.
- The combined size of the **InnoDB** log files must be less than 4GB.
- The minimum tablespace size is 10MB. The maximum tablespace size is four billion database pages (64TB). This is also the maximum size for a table.
- **InnoDB** tables do not support **FULLTEXT** indexes.
- **InnoDB** tables support spatial types, but not indexes on them.
- **ANALYZE TABLE** determines index cardinality (as displayed in the **Cardinality** column of **SHOW INDEX** output) by doing ten random dives to each of the index trees and updating index cardinality estimates accordingly. Note that because these are only estimates, repeated runs of **ANALYZE TABLE** may produce different numbers. This makes **ANALYZE TABLE** fast on **InnoDB** tables but not 100% accurate as it doesn't take all rows into account.

MySQL uses index cardinality estimates only in join optimization. If some join is not optimized in the right way, you can try using **ANALYZE TABLE**. In the few cases that **ANALYZE TABLE** doesn't produce values good enough for your particular tables, you can use **FORCE INDEX** with your queries to force the use of a particular index, or set the **max_seeks_for_key** system variable to ensure that MySQL prefers index lookups over table scans. See [Section 5.2.3, “System Variables”](#), and [Section B.1.6, “Optimizer-Related Issues”](#).

- **SHOW TABLE STATUS** does not give accurate statistics on **InnoDB** tables, except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimization.
- **InnoDB** does not keep an internal count of rows in a table. (In practice, this would be somewhat complicated due to multi-versioning.) To process a **SELECT COUNT(*) FROM t** statement, **InnoDB** must scan an index of the table, which takes some time if the index is not entirely in the buffer pool. To get a fast count, you have to use a counter table you create yourself and let your application update it according to the inserts and deletes it does. If your table does not change often, using the MySQL query cache is a good solution. **SHOW TABLE STATUS** also can be used if an approximate row count is sufficient. See [Section 13.5.11, “InnoDB Performance Tuning Tips”](#).
- On Windows, **InnoDB** always stores database and table names internally in lowercase. To move databases in binary format from Unix to Windows or from Windows to Unix, you should always use explicitly lowercase names when creating databases and tables.
- For an **AUTO_INCREMENT** column, you must always define an index for the table, and that index must contain just the **AUTO_INCREMENT** column. In **MyISAM** tables, the **AUTO_INCREMENT** column may be part of a multi-column index.
- While initializing a previously specified **AUTO_INCREMENT** column on a table, **InnoDB** sets an exclusive lock on the end of the index associated with the **AUTO_INCREMENT** column. In accessing the auto-increment counter, **InnoDB** uses a specific table lock mode **AUTO-INC** where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Note that other clients cannot insert into the table while the **AUTO-INC** table lock is held; see [Section 13.5.10.2, “InnoDB and AUTO-COMMIT”](#).
- When you restart the MySQL server, **InnoDB** may reuse an old value that was generated for an **AUTO_INCREMENT** column but never stored (that is, a value that was generated during an old transaction that was rolled back).
- When an **AUTO_INCREMENT** column runs out of values, **InnoDB** wraps a **BIGINT** to **-9223372036854775808** and **BIGINT UNSIGNED** to **1**. However, **BIGINT** values have 64 bits, so do note that if you were to insert one million rows per second, it would still take nearly three hundred thousand years before **BIGINT** reached its upper bound. With all other integer type columns, a duplicate-key error results. This is similar to how **MyISAM** works, because it is mostly general MySQL behavior and not about any storage engine in particular.
- **DELETE FROM tbl_name** does not regenerate the table but instead deletes all rows, one by one.
- Under some conditions, **TRUNCATE tbl_name** for an **InnoDB** table is mapped to **DELETE FROM tbl_name** and doesn't reset the **AUTO_INCREMENT** counter. See [Section 12.2.9, “TRUNCATE Syntax”](#).
- In MySQL 5.1, the MySQL **LOCK TABLES** operation acquires two locks on each table if **innodb_table_locks=1** (the default). In addition to a table lock on the MySQL layer, it also acquires an **InnoDB** table lock. Older versions of MySQL did not acquire **InnoDB** table locks; the old behavior can be selected by setting **innodb_table_locks=0**. If no **InnoDB** table lock is acquired, **LOCK TABLES** completes even if some records of the tables are being locked by other transactions.

- All **InnoDB** locks held by a transaction are released when the transaction is committed or aborted. Thus, it does not make much sense to invoke **LOCK TABLES** on **InnoDB** tables in **AUTOCOMMIT=1** mode, because the acquired **InnoDB** table locks would be released immediately.
- Sometimes it would be useful to lock further tables in the course of a transaction. Unfortunately, **LOCK TABLES** in MySQL performs an implicit **COMMIT** and **UNLOCK TABLES**. An **InnoDB** variant of **LOCK TABLES** has been planned that can be executed in the middle of a transaction.
- The **LOAD TABLE FROM MASTER** statement for setting up replication slave servers does not work for **InnoDB** tables. A work-around is to alter the table to **MyISAM** on the master, then do the load, and after that alter the master table back to **InnoDB**. Do not do this if the tables use **InnoDB**-specific features such as foreign keys.
- The default database page size in **InnoDB** is 16KB. By recompiling the code, you can set it to values ranging from 8KB to 64KB. You must update the values of **UNIV_PAGE_SIZE** and **UNIV_PAGE_SIZE_SHIFT** in the **univ.i** source file.
- Currently, triggers are not activated by cascaded foreign key actions.
- **InnoDB** has a limit of 1023 concurrent transactions that have created undo records by modifying data. Workarounds include keeping transactions as small and fast as possible, delaying changes until near the end of the transaction, and using stored routines to reduce client-server latency delays. Applications should commit transactions before doing time-consuming client-side operations.

13.5.17. InnoDB Troubleshooting

The following general guidelines apply to troubleshooting **InnoDB** problems:

- When an operation fails or you suspect a bug, you should look at the MySQL server error log, which is the file in the data directory that has a suffix of **.err**.
- When troubleshooting, it is usually best to run the MySQL server from the command prompt, rather than through the **mysqld_safe** wrapper or as a Windows service. You can then see what **mysqld** prints to the console, and so have a better grasp of what is going on. On Windows, you must start the server with the **--console** option to direct the output to the console window.
- Use the **InnoDB** Monitors to obtain information about a problem (see [Section 13.5.11.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”](#)). If the problem is performance-related, or your server appears to be hung, you should use **innodb_monitor** to print information about the internal state of **InnoDB**. If the problem is with locks, use **innodb_lock_monitor**. If the problem is in creation of tables or other data dictionary operations, use **innodb_table_monitor** to print the contents of the **InnoDB** internal data dictionary.
- If you suspect that a table is corrupt, run **CHECK TABLE** on that table.

MySQL Enterprise

The MySQL Enterprise Monitor provides a number of advisors specifically designed for monitoring **InnoDB** tables. In some cases, these advisors can anticipate potential problems. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

13.5.17.1. Troubleshooting InnoDB Data Dictionary Operations

A specific issue with tables is that the MySQL server keeps data dictionary information in **.frm** files it stores in the database directories, whereas **InnoDB** also stores the information into its own data dictionary inside the tablespace files. If you move **.frm** files around, or if the server crashes in the middle of a data dictionary operation, the locations of the **.frm** files may end up out of synchrony with the locations recorded in the **InnoDB** internal data dictionary.

A symptom of an out-of-sync data dictionary is that a **CREATE TABLE** statement fails. If this occurs, you should look in the server's error log. If the log says that the table already exists inside the **InnoDB** internal data dictionary, you have an orphaned table inside the **InnoDB** tablespace files that has no corresponding **.frm** file. The error message looks like this:

```
InnoDB: Error: table test/parent already exists in InnoDB internal
InnoDB: data dictionary. Have you deleted the .frm file
InnoDB: and not used DROP TABLE? Have you used DROP DATABASE
InnoDB: for InnoDB tables in MySQL version <= 3.23.43?
InnoDB: See the Restrictions section of the InnoDB manual.
InnoDB: You can drop the orphaned table inside InnoDB by
InnoDB: creating an InnoDB table with the same name in another
```

```
InnoDB: database and moving the .frm file to the current database.
InnoDB: Then MySQL thinks the table exists, and DROP TABLE will
InnoDB: succeed.
```

You can drop the orphaned table by following the instructions given in the error message. If you are still unable to use `DROP TABLE` successfully, the problem may be due to name completion in the `mysql` client. To work around this problem, start the `mysql` client with the `--skip-auto-rehash` option and try `DROP TABLE` again. (With name completion on, `mysql` tries to construct a list of table names, which fails when a problem such as just described exists.)

Another symptom of an out-of-sync data dictionary is that MySQL prints an error that it cannot open a `.InnoDB` file:

```
ERROR 1016: Can't open file: 'child2.InnoDB'. (errno: 1)
```

In the error log you can find a message like this:

```
InnoDB: Cannot find table test/child2 from the internal data dictionary
InnoDB: of InnoDB though the .frm file for the table exists. Maybe you
InnoDB: have deleted and recreated InnoDB data files but have forgotten
InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

This means that there is an orphaned `.frm` file without a corresponding table inside `InnoDB`. You can drop the orphaned `.frm` file by deleting it manually.

If MySQL crashes in the middle of an `ALTER TABLE` operation, you may end up with an orphaned temporary table inside the `InnoDB` tablespace. Using `innodb_table_monitor` you can see listed a table whose name is `#sql-...`. You can perform SQL statements on tables whose name contains the character `#` if you enclose the name within backticks. Thus, you can drop such an orphaned table like any other orphaned table using the method described earlier. Note that to copy or rename a file in the Unix shell, you need to put the file name in double quotes if the file name contains `#`.

13.6. The MERGE Storage Engine

The `MERGE` storage engine, also known as the `MRG_MyISAM` engine, is a collection of identical `MyISAM` tables that can be used as one. “Identical” means that all tables have identical column and index information. You cannot merge `MyISAM` tables in which the columns are listed in a different order, do not have exactly the same columns, or have the indexes in different order. However, any or all of the `MyISAM` tables can be compressed with `myisampack`. See [Section 7.7, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#). Differences in table options such as `AVG_ROW_LENGTH`, `MAX_ROWS`, or `PACK_KEYS` do not matter.

When you create a `MERGE` table, MySQL creates two files on disk. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table format, and an `.MRG` file contains the names of the tables that should be used as one. The tables do not have to be in the same database as the `MERGE` table itself.

You can use `SELECT`, `DELETE`, `UPDATE`, and `INSERT` on `MERGE` tables. You must have `SELECT`, `UPDATE`, and `DELETE` privileges on the `MyISAM` tables that you map to a `MERGE` table.

Note

The use of `MERGE` tables entails the following security issue: If a user has access to `MyISAM` table `t`, that user can create a `MERGE` table `m` that accesses `t`. However, if the user's privileges on `t` are subsequently revoked, the user can continue to access `t` by doing so through `m`. If this behavior is undesirable, you can start the server with the new `--skip-merge` option to disable the `MERGE` storage engine. This option is available as of MySQL 5.1.12.

If you `DROP` the `MERGE` table, you are dropping only the `MERGE` specification. The underlying tables are not affected.

To create a `MERGE` table, you must specify a `UNION=(list-of-tables)` clause that indicates which `MyISAM` tables you want to use as one. You can optionally specify an `INSERT_METHOD` option if you want inserts for the `MERGE` table to take place in the first or last table of the `UNION` list. Use a value of `FIRST` or `LAST` to cause inserts to be made in the first or last table, respectively. If you do not specify an `INSERT_METHOD` option or if you specify it with a value of `NO`, attempts to insert rows into the `MERGE` table result in an error.

The following example shows how to create a `MERGE` table:

```
mysql> CREATE TABLE t1 (
->   a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
->   message CHAR(20)) ENGINE=MyISAM;
mysql> CREATE TABLE t2 (
->   a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
```

```

-> message CHAR(20)) ENGINE=MyISAM;
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
->   a INT NOT NULL AUTO_INCREMENT,
->   message CHAR(20), INDEX(a))
->   ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;

```

Note that the `a` column is indexed as a **PRIMARY KEY** in the underlying **MyISAM** tables, but not in the **MERGE** table. There it is indexed but not as a **PRIMARY KEY** because a **MERGE** table cannot enforce uniqueness over the set of underlying tables.

In MySQL 5.1.15 and later, when a table that is part of a **MERGE** table is opened, the following checks are applied before opening each table. If any table fails the conformance checks, then the operation that triggered the opening of the table will fail. The conformance checks applied to each table are:

- Table must have exactly the same amount of columns that **MERGE** table has.
- Column order in the **MERGE** table must match the column order in the underlying tables.
- Additionally, the specification for each column in the parent **MERGE** table and the underlying table are compared. For each column, MySQL checks:
 - Column type in the underlying table equals the column type of **MERGE** table.
 - Column length in the underlying table equals the column length of **MERGE** table.
 - Column of underlying table and column of **MERGE** table can be **NULL**.
- Underlying table must have at least the same amount of keys that merge table has. The underlying table may have more keys than the **MERGE** table, but cannot have less.
- For each key:
 - Check if the key type of underlying table equals the key type of merge table.
 - Check if number of key parts (i.e. multiple columns within a compound key) in the underlying table key definition equals the number of key parts in merge table key definition.
 - For each key part:
 - Check if key part lengths are equal.
 - Check if key part types are equal.
 - Check if key part languages are equal.
 - Check if key part can be **NULL**.

After creating the **MERGE** table, you can issue queries that operate on the group of tables as a whole:

```

mysql> SELECT * FROM total;
+-----+-----+
| a | message |
+-----+-----+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+-----+-----+

```

To remap a **MERGE** table to a different collection of **MyISAM** tables, you can use one of the following methods:

- **DROP** the **MERGE** table and re-create it.
- Use **ALTER TABLE *tbl_name* UNION=(...)** to change the list of underlying tables.

MERGE tables can help you solve the following problems:

- Easily manage a set of log tables. For example, you can put data from different months into separate tables, compress some of them with **myisampack**, and then create a **MERGE** table to use them as one.
- Obtain more speed. You can split a big read-only table based on some criteria, and then put individual tables on different disks. A **MERGE** table on this could be much faster than using the big table.
- Perform more efficient searches. If you know exactly what you are looking for, you can search in just one of the split tables for some queries and use a **MERGE** table for others. You can even have many different **MERGE** tables that use overlapping sets of tables.
- Perform more efficient repairs. It is easier to repair individual tables that are mapped to a **MERGE** table than to repair a single large table.
- Instantly map many tables as one. A **MERGE** table need not maintain an index of its own because it uses the indexes of the individual tables. As a result, **MERGE** table collections are very fast to create or remap. (Note that you must still specify the index definitions when you create a **MERGE** table, even though no indexes are created.)
- If you have a set of tables from which you create a large table on demand, you should instead create a **MERGE** table on them on demand. This is much faster and saves a lot of disk space.
- Exceed the file size limit for the operating system. Each **MyISAM** table is bound by this limit, but a collection of **MyISAM** tables is not.
- You can create an alias or synonym for a **MyISAM** table by defining a **MERGE** table that maps to that single table. There should be no really notable performance impact from doing this (only a couple of indirect calls and **memcpy()** calls for each read).

The disadvantages of **MERGE** tables are:

- You can use only identical **MyISAM** tables for a **MERGE** table.
- You cannot use a number of **MyISAM** features in **MERGE** tables. For example, you cannot create **FULLTEXT** indexes on **MERGE** tables. (You can, of course, create **FULLTEXT** indexes on the underlying **MyISAM** tables, but you cannot search the **MERGE** table with a full-text search.)
- If the **MERGE** table is non-temporary, all underlying **MyISAM** tables must be non-temporary, too. If the **MERGE** table is temporary, the **MyISAM** tables can be any mix of temporary and non-temporary.
- **MERGE** tables use more file descriptors. If 10 clients are using a **MERGE** table that maps to 10 tables, the server uses $(10 \times 10) + 10$ file descriptors. (10 data file descriptors for each of the 10 clients, and 10 index file descriptors shared among the clients.)
- Key reads are slower. When you read a key, the **MERGE** storage engine needs to issue a read on all underlying tables to check which one most closely matches the given key. To read the next key, the **MERGE** storage engine needs to search the read buffers to find the next key. Only when one key buffer is used up does the storage engine need to read the next key block. This makes **MERGE** keys much slower on **eq_ref** searches, but not much slower on **ref** searches. See [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#), for more information about **eq_ref** and **ref**.

Additional resources

- A forum dedicated to the **MERGE** storage engine is available at <http://forums.mysql.com/list.php?93>.

13.6.1. **MERGE** Table Problems

The following are known problems with **MERGE** tables:

- If you use **ALTER TABLE** to change a **MERGE** table to another storage engine, the mapping to the underlying tables is lost. Instead, the rows from the underlying **MyISAM** tables are copied into the altered table, which then uses the specified storage engine.
- **REPLACE** does not work.

- **MERGE** tables do not support partitioning. That is, you cannot partition a **MERGE** table, nor can any of a **MERGE** table's underlying **MyISAM** tables be partitioned.
- You cannot use **REPAIR TABLE**, **OPTIMIZE TABLE**, **DROP TABLE**, **ALTER TABLE**, **DELETE** without a **WHERE** clause, **TRUNCATE TABLE**, or **ANALYZE TABLE** on any of the tables that are mapped into an open **MERGE** table. If you do so, the **MERGE** table may still refer to the original table, which yields unexpected results. The easiest way to work around this deficiency is to ensure that no **MERGE** tables remain open by issuing a **FLUSH TABLES** statement prior to performing any of those operations.

The unexpected results include the possibility that the operation on the **MERGE** table will report table corruption. However, if this occurs after operations on the underlying **MyISAM** tables such as those listed in the previous paragraph (**REPAIR TABLE**, **OPTIMIZE TABLE**, and so forth), the corruption message is spurious. To deal with this, issue a **FLUSH TABLES** statement after modifying the **MyISAM** tables.

- **DROP TABLE** on a table that is in use by a **MERGE** table does not work on Windows because the **MERGE** storage engine's table mapping is hidden from the upper layer of MySQL. Windows does not allow open files to be deleted, so you first must flush all **MERGE** tables (with **FLUSH TABLES**) or drop the **MERGE** table before dropping the table.
- A **MERGE** table cannot maintain uniqueness constraints over the entire table. When you perform an **INSERT**, the data goes into the first or last **MyISAM** table (depending on the value of the **INSERT_METHOD** option). MySQL ensures that unique key values remain unique within that **MyISAM** table, but not across all the tables in the collection.
- In MySQL 5.1.15 and later, the definition of the **MyISAM** tables and the **MERGE** table are checked when the tables are accessed (for example, as part of a **SELECT** or **INSERT** statement). The checks ensure that the definitions of the tables and the parent **MERGE** table definition match by comparing column order, types, sizes and associated indexes. If there is a difference between the tables then an error will be returned and the statement will fail.

Because these checks take place when the tables are opened, any changes to the definition of a single, including column changes, column ordering and engine alterations will cause the statement to fail.

In MySQL 5.1.14 and earlier:

- When you create or alter **MERGE** table, there is no check to ensure that the underlying tables are existing **MyISAM** tables and have identical structures. When the **MERGE** table is used, MySQL checks that the row length for all mapped tables is equal, but this is not foolproof. If you create a **MERGE** table from dissimilar **MyISAM** tables, you are very likely to run into strange problems.
- Similarly, if you create a **MERGE** table from non-**MyISAM** tables, or if you drop an underlying table or alter it to be a non-**MyISAM** table, no error for the **MERGE** table occurs until later when you attempt to use it.
- Because the underlying **MyISAM** tables need not exist when the **MERGE** table is created, you can create the tables in any order, as long as you do not use the **MERGE** table until all of its underlying tables are in place. Also, if you can ensure that a **MERGE** table will not be used during a given period, you can perform maintenance operations on the underlying tables, such as backing up or restoring them, altering them, or dropping and recreating them. It is not necessary to redefine the **MERGE** table temporarily to exclude the underlying tables while you are operating on them.
- The order of indexes in the **MERGE** table and its underlying tables should be the same. If you use **ALTER TABLE** to add a **UNIQUE** index to a table used in a **MERGE** table, and then use **ALTER TABLE** to add a non-unique index on the **MERGE** table, the index ordering is different for the tables if there was already a non-unique index in the underlying table. (This happens because **ALTER TABLE** puts **UNIQUE** indexes before non-unique indexes to facilitate rapid detection of duplicate keys.) Consequently, queries on tables with such indexes may return unexpected results.
- If you encounter an error message similar to **ERROR 1017 (HY000): Can't find file: 'mm.MRG' (errno: 2)** it generally indicates that some of the base tables are not using the **MyISAM** storage engine. Confirm that all tables are **MyISAM**.
- There is a limit of 2^{32} (~4.295E+09) rows to a **MERGE** table, just as there is with a **MyISAM**, it is therefore not possible to merge multiple **MyISAM** tables that exceed this limitation. However, you build MySQL with the **--with-big-tables** option then the row limitation is increased to $(2^{32})^2$ (1.844E+19) rows. See [Section 2.9.2, "Typical configure Options"](#). Beginning with MySQL 5.0.4 all standard binaries are built with this option.
- The **MERGE** storage engine does not support **INSERT DELAYED** statements.

As of MySQL 5.1.20, if a **MERGE** table cannot be opened or used because of a problem with an underlying table, **CHECK TABLE** displays information about which table caused the problem.

13.7. The MEMORY (HEAP) Storage Engine

The **MEMORY** storage engine creates tables with contents that are stored in memory. Formerly, these were known as **HEAP** tables. **MEMORY** is the preferred term, although **HEAP** remains supported for backward compatibility.

Each **MEMORY** table is associated with one disk file. The filename begins with the table name and has an extension of **.frm** to indicate that it stores the table definition.

To specify explicitly that you want to create a **MEMORY** table, indicate that with an **ENGINE** table option:

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

As indicated by the name, **MEMORY** tables are stored in memory. They use hash indexes by default, which makes them very fast, and very useful for creating temporary tables. However, when the server shuts down, all rows stored in **MEMORY** tables are lost. The tables themselves continue to exist because their definitions are stored in **.frm** files on disk, but they are empty when the server restarts.

This example shows how you might create, use, and remove a **MEMORY** table:

```
mysql> CREATE TABLE test ENGINE=MEMORY
-> SELECT ip,SUM(downloads) AS down
-> FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

MEMORY tables have the following characteristics:

- Space for **MEMORY** tables is allocated in small blocks. Tables use 100% dynamic hashing for inserts. No overflow area or extra key space is needed. No extra space is needed for free lists. Deleted rows are put in a linked list and are reused when you insert new data into the table. **MEMORY** tables also have none of the problems commonly associated with deletes plus inserts in hashed tables.
- **MEMORY** tables can have up to 32 indexes per table, 16 columns per index and a maximum key length of 500 bytes.
- The **MEMORY** storage engine implements both **HASH** and **BTREE** indexes. You can specify one or the other for a given index by adding a **USING** clause as shown here:

```
CREATE TABLE lookup
(id INT, INDEX USING HASH (id))
ENGINE = MEMORY;
CREATE TABLE lookup
(id INT, INDEX USING BTREE (id))
ENGINE = MEMORY;
```

General characteristics of B-tree and hash indexes are described in [Section 6.4.5, “How MySQL Uses Indexes”](#).

- You can have non-unique keys in a **MEMORY** table. (This is an uncommon feature for implementations of hash indexes.)
- If you have a hash index on a **MEMORY** table that has a high degree of key duplication (many index entries containing the same value), updates to the table that affect key values and all deletes are significantly slower. The degree of this slowdown is proportional to the degree of duplication (or, inversely proportional to the index cardinality). You can use a **BTREE** index to avoid this problem.
- Columns that are indexed can contain **NULL** values.
- **MEMORY** tables use a fixed-length row storage format.
- **MEMORY** tables cannot contain **BLOB** or **TEXT** columns.
- **MEMORY** includes support for **AUTO_INCREMENT** columns.
- You can use **INSERT DELAYED** with **MEMORY** tables. See [Section 12.2.4.2, “INSERT DELAYED Syntax”](#).
- **MEMORY** tables are shared among all clients (just like any other non-**TEMPORARY** table).
- **MEMORY** table contents are stored in memory, which is a property that **MEMORY** tables share with internal tables that the server creates on the fly while processing queries. However, the two types of tables differ in that **MEMORY** tables are not subject to storage conversion, whereas internal tables are:

- If an internal table becomes too large, the server automatically converts it to an on-disk table. The size limit is determined by the value of the `tmp_table_size` system variable.
- `MEMORY` tables are never converted to disk tables. To ensure that you don't accidentally do anything foolish, you can set the `max_heap_table_size` system variable to impose a maximum size on `MEMORY` tables. For individual tables, you can also specify a `MAX_ROWS` table option in the `CREATE TABLE` statement.
- The server needs sufficient memory to maintain all `MEMORY` tables that are in use at the same time.
- Memory used by a `MEMORY` table is not reclaimed if you delete individual rows from the table. Memory is only reclaimed when the entire table is deleted. Memory that was previously used for rows that have been deleted will be re-used for new rows only within the same table. To free up the memory used by rows that have been deleted you should use `ALTER TABLE ENGINE=MEMORY` to force a table rebuild.

To free all the memory used by a `MEMORY` table when you no longer require its contents, you should execute `DELETE` or `TRUNCATE TABLE`, or remove the table altogether using `DROP TABLE`.

- If you want to populate a `MEMORY` table when the MySQL server starts, you can use the `--init-file` option. For example, you can put statements such as `INSERT INTO ... SELECT` or `LOAD DATA INFILE` into this file to load the table from a persistent data source. See [Section 5.2.2, “Command Options”](#), and [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).
- If you are using replication, the master server's `MEMORY` tables become empty when it is shut down and restarted. However, a slave is not aware that these tables have become empty, so it returns out-of-date content if you select data from them. When a `MEMORY` table is used on the master for the first time since the master was started, a `DELETE` statement is written to the master's binary log automatically, thus synchronizing the slave to the master again. Note that even with this strategy, the slave still has outdated data in the table during the interval between the master's restart and its first use of the table. However, if you use the `--init-file` option to populate the `MEMORY` table on the master at startup, it ensures that this time interval is zero.
- The memory needed for one row in a `MEMORY` table is calculated using the following expression:

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) × 4)
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) × 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

`ALIGN()` represents a round-up factor to cause the row length to be an exact multiple of the `char` pointer size. `sizeof(char*)` is 4 on 32-bit machines and 8 on 64-bit machines.

Additional resources

- A forum dedicated to the `MEMORY` storage engine is available at <http://forums.mysql.com/list.php?92>.

13.8. The `EXAMPLE` Storage Engine

The `EXAMPLE` storage engine is a stub engine that does nothing. Its purpose is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

To enable the `EXAMPLE` storage engine if you build MySQL from source, invoke `configure` with the `-with-example-storage-engine` option.

To examine the source for the `EXAMPLE` engine, look in the `storage/example` directory of a MySQL source distribution.

When you create an `EXAMPLE` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. No other files are created. No data can be stored into the table. Retrievals return an empty result.

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't »
have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```


The **EXAMPLE** storage engine does not support indexing.

13.9. The **FEDERATED** Storage Engine

The **FEDERATED** storage engine enables data to be accessed from a remote MySQL database on a local server without using replication or cluster technology. When using a **FEDERATED** table, queries on the local server are automatically executed on the remote (federated) tables. No data is stored on the local tables.

To enable the **FEDERATED** storage engine if you build MySQL from source, invoke **configure** with the **-with-federated-storage-engine** option.

To examine the source for the **FEDERATED** engine, look in the **sql** directory of a MySQL source distribution.

MySQL Enterprise

MySQL Enterprise subscribers will find MySQL Knowledge Base articles about the **FEDERATED** storage engine at **FEDERATED Storage Engine**. Access to the Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

13.9.1. **FEDERATED** Storage Engine Overview

When you create a table using one of the standard storage engines (such as **MyISAM**, **CSV** or **InnoDB**, the table consists of the table definition and the associated data. When you create a **FEDERATED** table the table definition is the same, but the physical storage of the data is handled on a remote server.

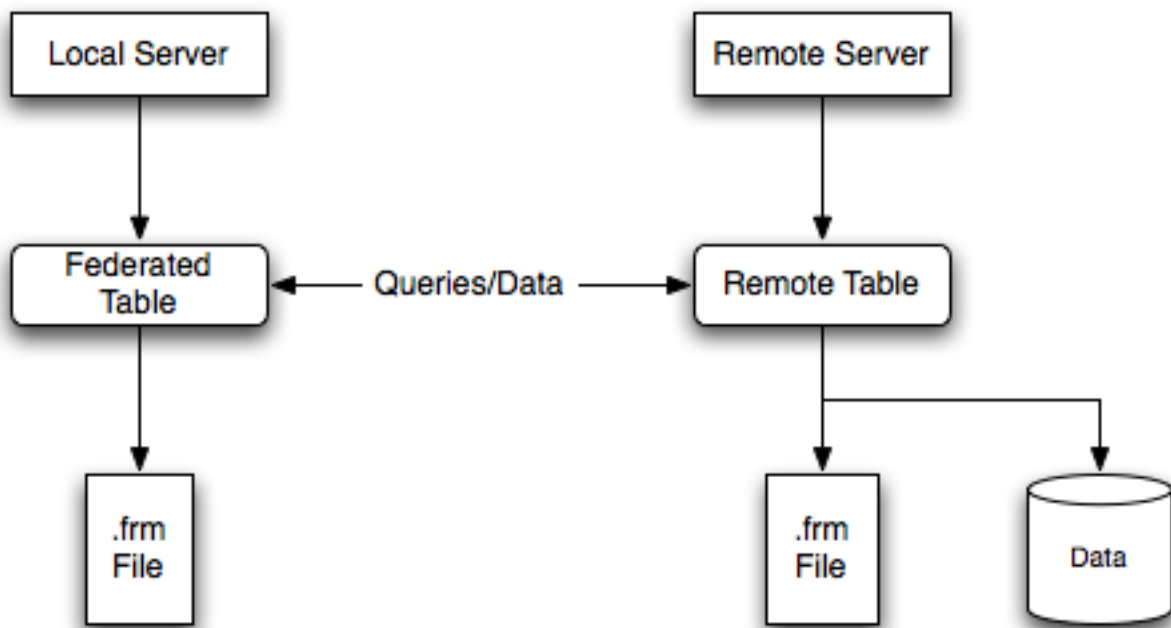
A **FEDERATED** table consists of two elements:

- A *remote server* with a database table, which in turn consists of the table definition (stored in the **.frm** file) and the associated table. The table type of the remote table may be any type supported by the remote **mysqld** server, including **MyISAM** or **InnoDB**.
- A *local server* with a database table, where the table definition matches that of the corresponding table on the remote server. The table definition is stored within the **.frm** file. However, there is no data file on the local server. Instead, the table definition includes a connection string that points to the remote table.

When executing queries and statements on a **FEDERATED** table on the local server, the operations that would normally insert, update or delete information from a local data file are instead sent to the remote server for execution, where they update the data file on the remote server or return matching rows from the remote server.

The basic structure of a **FEDERATED** table set up is shown in **Figure 13.2, “FEDERATED table structure”**.

Figure 13.2. **FEDERATED table structure**



When a client issues a SQL statement that refers to a **FEDERATED** table, the flow of information between the local server (where the SQL statement is executed) and the remote server (where the data is physically stored) is as follows:

1. The storage engine looks through each column that the **FEDERATED** table has and constructs an appropriate SQL statement that refers to the remote table.
2. The statement is sent to the remote server using the MySQL client API.
3. The remote server processes the statement and the local server retrieves any result that the statement produces (an affected-rows count or a result set).
4. If the statement produces a result set, each column is converted to internal storage engine format that the **FEDERATED** engine expects and can use to display the result to the client that issued the original statement.

The local server communicates with the remote server using MySQL client C API functions. It invokes `mysql_real_query()` to send the statement. To read a result set, it uses `mysql_store_result()` and fetches rows one at a time using `mysql_fetch_row()`.

13.9.2. How to create **FEDERATED** Tables

To create a **FEDERATED** table you should follow these steps:

1. Create the table on the remote server. Alternatively, make a note of the table definition of an existing table, perhaps using the `SHOW CREATE TABLE` statement.
2. Create the table on the local server with an identical table definition, but adding the connection information that links the local table to the remote table.

For example, you could create the following table on the remote server:

```
CREATE TABLE test_table (
  id      INT(20) NOT NULL AUTO_INCREMENT,
  name    VARCHAR(32) NOT NULL DEFAULT '',
  other   INT(20) NOT NULL DEFAULT '0',
```

```

PRIMARY KEY (id),
INDEX name (name),
INDEX other_key (other)
)
ENGINE=MyISAM
DEFAULT CHARSET=latin1;

```

To create the local table that will be federated to the remote table, there are two options available. You can either create the local table and specify the connection string (containing the server name, login, password) to be used to connect to the remote table using the [CONNECTION](#), or you can use an existing connection that you have previously created using the [CREATE SERVER](#) statement.

Note

When you create the local table is *must* have an identical definition to the remote table.

13.9.2.1. Creating a **FEDERATED** Table Using **CONNECTION**

To use the first method, you must specify the [CONNECTION](#) string after the engine type in a [CREATE TABLE](#) statement. For example:

```

CREATE TABLE federated_table (
  id      INT(20) NOT NULL AUTO_INCREMENT,
  name    VARCHAR(32) NOT NULL DEFAULT '',
  other   INT(20) NOT NULL DEFAULT '0',
  PRIMARY KEY (id),
  INDEX name (name),
  INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';

```

Note

[CONNECTION](#) replaces the [COMMENT](#) used in some previous versions of MySQL.

The [CONNECTION](#) string contains the information required to connect to the remote server containing the table that will be used to physically store the data. The connection string specifies the server name, login credentials, port number and database/table information. In the example, the remote table is on the server [remote_host](#), using port 9306. The name and port number should match the host-name (or IP address) and port number of the remote MySQL server instance you want to use as your remote table.

The format the connection string is as follows:

```

scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name

```

Where:

- [scheme](#) — is a recognized connection protocol. Only [mysql](#) is supported as the [scheme](#) value at this point.
- [user_name](#) — the user name for the connection. This user must have been created on the remote server, and must have suitable privileges to perform the required actions ([SELECT](#), [INSERT](#), [UPDATE](#) etc.) on the remote table.
- [password](#) — (optional) the corresponding password for [username](#).
- [host_name](#) — the hostname or IP address of the remote server.
- [port_num](#) — (optional) the port number for the remote server. The default is 3306.
- [db_name](#) — the name of the database holding the remote table.
- [table_name](#) — the name of the remote table. The name of the local and the remote table do not have to match.

Sample connection strings:

```

CONNECTION='mysql://username:password@hostname:port/database/tablename'
CONNECTION='mysql://username@hostname/database/tablename'
CONNECTION='mysql://username:password@hostname/database/tablename'

```

13.9.2.2. Creating a **FEDERATED** Table Using **CREATE SERVER**

If you are creating a number of **FEDERATED** tables on the same server, or if you want to simplify the process of creating **FEDERATED** tables, then you can use the **CREATE SERVER** statement to define the server connection parameters, just as you would with the **CONNECTION** string.

The format of the **CREATE SERVER** statement is:

```
CREATE SERVER
  server_name
  FOREIGN DATA WRAPPER wrapper_name
  OPTIONS (option ...)
```

The `server_name` is used as the connection string when creating a new **FEDERATED** table.

For example, to create a server connection identical to the **CONNECTION** string:

```
CONNECTION='mysql://root@remote_host:9306/federated/test_table';
```

You would use the following statement:

```
CREATE SERVER fedlink
  FOREIGN DATA WRAPPER mysql
  OPTIONS (USER 'fed_user', HOST 'remote_host', PORT 9306, DATABASE 'federated');
```

To create a **FEDERATED** table that uses this connection, you still use the **CONNECTION** keyword, but specify the name you used in the **CREATE SERVER** statement.

```
CREATE TABLE test_table (
  id      INT(20) NOT NULL AUTO_INCREMENT,
  name    VARCHAR(32) NOT NULL DEFAULT '',
  other   INT(20) NOT NULL DEFAULT '0',
  PRIMARY KEY (id),
  INDEX name (name),
  INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='fedlink.test_table';
```

The connection name in this example contains the name of the connection (`fedlink`) and the name of the table (`test_table`) to link to, separated by a period. If you do not explicitly specify a table name in this manner, then the table name of the local table is used instead.

For more information on **CREATE SERVER**, see [Section 12.1.9, “CREATE SERVER Syntax”](#).

The **CREATE SERVER** statement accepts the same arguments as the **CONNECTION** string. The **CREATE SERVER** statement updates the rows in the `mysql.servers` table. See the table for information on the parameters when using the connection string, options in the **CREATE SERVER** statement, and the columns in the `mysql.servers` table. For reference, the format of the **CONNECTION** string is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Description	CONNECTION string	CREATE SERVER option	<code>mysql.servers</code> column
Connection scheme	<code>scheme</code>	<code>wrapper_name</code>	<code>Wrapper</code>
Remote user	<code>user_name</code>	<code>USER</code>	<code>Username</code>
Remote password	<code>password</code>	<code>PASSWORD</code>	<code>Password</code>
Remote host	<code>host_name</code>	<code>HOST</code>	<code>Host</code>
Remote port	<code>port_num</code>	<code>PORT</code>	<code>Port</code>
Remote database	<code>db_name</code>	<code>DATABASE</code>	<code>Db</code>

13.9.3. **FEDERATED** Storage Engine Notes and Tips

You should be aware of the following points when using the **FEDERATED** storage engine:

- **FEDERATED** tables may be replicated to other slaves, but you must ensure that the slave servers are able to use the user/password combination that is defined in the **CONNECTION** (or the row in the `mysql.servers` table) to connect to the remote server.

The following items indicate features that the **FEDERATED** storage engine does and does not support:

- The remote server must be a MySQL server. Support by **FEDERATED** for other database engines may be added in the future.
- The remote table that a **FEDERATED** table points to *must* exist before you try to access the table through the **FEDERATED** table.
- It is possible for one **FEDERATED** table to point to another, but you must be careful not to create a loop.
- A **FEDERATED** table does not support indexes per se. Because access to the table is handled remotely, it is the remote table that supports the indexes. Care should be taken when creating a **FEDERATED** table since the index definition from an equivalent **MyISAM** or other table may not be supported. For example, creating a **FEDERATED** table with an index prefix on **VARCHAR**, **TEXT** or **BLOB** columns will fail. The following definition in **MyISAM** is valid:

```
CREATE TABLE `T1` (`A` VARCHAR(100), UNIQUE KEY(`A` (30))) ENGINE=MYISAM;
```

The key prefix in this example is incompatible with the **FEDERATED** engine, and the equivalent statement will fail:

```
CREATE TABLE `T1` (`A` VARCHAR(100), UNIQUE KEY(`A` (30))) ENGINE=FEDERATED
CONNECTION='MYSQL://127.0.0.1:3306/TEST/T1';
```

If possible, you should try to separate the column and index definition when creating tables on both the remote server and the local server to avoid these index issues.

- Internally, implementation uses **SELECT**, **INSERT**, **UPDATE**, and **DELETE**, but not **HANDLER**.
- The **FEDERATED** storage engine supports **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **TRUNCATE**, and indexes. It does not support **ALTER TABLE**, or any Data Definition Language statements other than **DROP TABLE**. The current implementation does not use prepared statements.
- **FEDERATED** accepts **INSERT ... ON DUPLICATE KEY UPDATE** statements, but if a duplicate-key violation occurs, the statement fails with an error.
- Performance on a **FEDERATED** table when performing bulk inserts (for example, on a **INSERT INTO ... SELECT ...** statement) is slower than with other table types because each selected row is treated as an individual **INSERT** statement on the **FEDERATED** table.
- Transactions are not supported.
- Before MySQL 5.1.21, for a multiple-row insert into a **FEDERATED** table that refers to a remote transactional table, if the insert failed for a row due to constraint failure, the remote table would contain a partial commit (the rows preceding the failed one) instead of rolling back the statement completely. This occurred because the rows were treated as individual inserts.

As of MySQL 5.1.21, **FEDERATED** performs bulk-insert handling such that multiple rows are sent to the remote table in a batch. This provides a performance improvement and enables the remote table to perform improvement. Also, if the remote table is transactional, it enables the remote storage engine to perform statement rollback properly should an error occur. This capability has the following limitations:

- The size of the insert cannot exceed the maximum packet size between servers. If the insert exceeds this size, it is broken into multiple packets and the rollback problem can occur.
- Bulk-insert handling does not occur for **INSERT ... ON DUPLICATE KEY UPDATE**.
- There is no way for the **FEDERATED** engine to know if the remote table has changed. The reason for this is that this table must work like a data file that would never be written to by anything other than the database system. The integrity of the data in the local table could be breached if there was any change to the remote database.
- When using a **CONNECTION** string, you cannot use an '@' character in the password. You can get round this limitation by using the

`CREATE SERVER` statement to create a server connection.

- The `INSERT_ID` and `TIMESTAMP` options are not propagated to the data provider.
- Any `DROP TABLE` statement issued against a `FEDERATED` table will drop only the local table, not the remote table.
- `FEDERATED` tables do not work with the query cache.
- User-defined partitioning is not supported for `FEDERATED` tables. Beginning with MySQL 5.1.15, it is no longer possible to create such tables at all.

Some of these limitations may be lifted in future versions of the `FEDERATED` handler.

13.9.4. `FEDERATED` Storage Engine Resources

The following additional resources are available for the `FEDERATED` storage engine:

- A forum dedicated to the `FEDERATED` storage engine is available at <http://forums.mysql.com/list.php?105>.

13.10. The `ARCHIVE` Storage Engine

The `ARCHIVE` storage engine is used for storing large amounts of data without indexes in a very small footprint.

The `ARCHIVE` storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke `configure` with the `--with-archive-storage-engine` option.

To examine the source for the `ARCHIVE` engine, look in the `storage/archive` directory of a MySQL source distribution.

You can check whether the `ARCHIVE` storage engine is available with this statement:

```
mysql> SHOW VARIABLES LIKE 'have_archive';
```

When you create an `ARCHIVE` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. The storage engine creates other files, all having names beginning with the table name. The data and metadata files have extensions of `.ARZ` and `.ARM`, respectively. An `.ARN` file may appear during optimization operations.

The `ARCHIVE` engine supports `INSERT` and `SELECT`, but not `DELETE`, `REPLACE`, or `UPDATE`. It does support `ORDER BY` operations, `BLOB` columns, and basically all but spatial data types (see [Section 18.4.1, “MySQL Spatial Data Types”](#)). The `ARCHIVE` engine uses row-level locking.

As of MySQL 5.1.6, the `ARCHIVE` engine supports the `AUTO_INCREMENT` column attribute. The `AUTO_INCREMENT` columns can have either a unique or non-unique index. Attempting to create an index on any other column results in an error. The `ARCHIVE` engine also supports the `AUTO_INCREMENT` table option in `CREATE TABLE` and `ALTER TABLE` statements to specify the initial sequence value for a new table or reset the sequence value for an existing table, respectively.

As of MySQL 5.1.6, the `ARCHIVE` engine ignores `BLOB` columns if they are not requested and scans past them while reading. Formerly, the following two statements had the same cost, but as of 5.1.6, the second is much more efficient than the first:

```
SELECT a, b, blob_col FROM archive_table;  
SELECT a, b FROM archive_table;
```

Storage: Rows are compressed as they are inserted. The `ARCHIVE` engine uses `zlib` lossless data compression (see <http://www.zlib.net/>). You can use `OPTIMIZE TABLE` to analyze the table and pack it into a smaller format (for a reason to use `OPTIMIZE TABLE`, see later in this section). The engine also supports `CHECK TABLE`. There are several types of insertions that are used:

- An `INSERT` statement just pushes rows into a compression buffer, and that buffer flushes as necessary. The insertion into the buffer is protected by a lock. A `SELECT` forces a flush to occur, unless the only insertions that have come in were `INSERT DELAYED` (those flush as necessary). See [Section 12.2.4.2, “INSERT DELAYED Syntax”](#).

- A bulk insert is visible only after it completes, unless other inserts occur at the same time, in which case it can be seen partially. A `SELECT` never causes a flush of a bulk insert unless a normal insert occurs while it is loading.

Retrieval: On retrieval, rows are uncompressed on demand; there is no row cache. A `SELECT` operation performs a complete table scan: When a `SELECT` occurs, it finds out how many rows are currently available and reads that number of rows. `SELECT` is performed as a consistent read. Note that lots of `SELECT` statements during insertion can deteriorate the compression, unless only bulk or delayed inserts are used. To achieve better compression, you can use `OPTIMIZE TABLE` or `REPAIR TABLE`. The number of rows in `ARCHIVE` tables reported by `SHOW TABLE STATUS` is always accurate. See [Section 12.5.2.5, “OPTIMIZE TABLE Syntax”](#), [Section 12.5.2.6, “REPAIR TABLE Syntax”](#), and [Section 12.5.4.28, “SHOW TABLE STATUS Syntax”](#).

Additional resources

- A forum dedicated to the `ARCHIVE` storage engine is available at <http://forums.mysql.com/list.php?112>.

13.11. The CSV Storage Engine

The `CSV` storage engine stores data in text files using comma-separated values format.

To enable the `CSV` storage engine if you build MySQL from source, invoke `configure` with the `-with-csv-storage-engine` option.

To examine the source for the `CSV` engine, look in the `storage/csv` directory of a MySQL source distribution.

When you create a `CSV` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. The storage engine also creates a data file. Its name begins with the table name and has a `.CSV` extension. The data file is a plain text file. When you store data into the table, the storage engine saves it into the data file in comma-separated values format.

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = CSV;
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM test;
+----+-----+
| i  | c      |
+----+-----+
| 1  | record one |
| 2  | record two |
+----+-----+
2 rows in set (0.00 sec)
```

Starting with MySQL 5.1.9, creating a CSV table also creates a corresponding Meta-file that stores the state of the table and the number of rows that exist in the table. The name of this file is the same as the name of the table with the extension `CSM`.

If you examine the `test.CSV` file in the database directory created by executing the preceding statements, its contents should look like this:

```
"1","record one"
"2","record two"
```

This format can be read, and even written, by spreadsheet applications such as Microsoft Excel or StarOffice Calc.

13.11.1. Repairing and Checking CSV Tables

Functionality introduced in version 5.1.9

The `CSV` storage engines supports the `CHECK` and `REPAIR` commands to verify and if possible repair a damaged `CSV` table.

When running the `CHECK` command, the `CSV` file will be checked for validity by looking for the correct field separators, escaped fields (matching quotes and/or missing quotes), the correct number of fields compared to the table definition and the existence of a corresponding `CSV` metafile. The first invalid row discovered will report an error. Checking a valid table produces output like that shown below:

```
mysql> check table csvtest;
```

Table	Op	Msg_type	Msg_text
test.csvtest	check	status	OK

```
1 row in set (0.00 sec)
```

A check on a corrupted table returns a fault:

```
mysql> check table csvtest;
```

Table	Op	Msg_type	Msg_text
test.csvtest	check	error	Corrupt

```
1 row in set (0.01 sec)
```

If the check fails, the table is marked as crashed (corrupt). Once a table has been marked as corrupt, it is automatically repaired when you next run **CHECK** or execute a **SELECT** statement. The corresponding corrupt status and new status will be displayed when running **CHECK**:

```
mysql> check table csvtest;
```

Table	Op	Msg_type	Msg_text
test.csvtest	check	warning	Table is marked as crashed
test.csvtest	check	status	OK

```
2 rows in set (0.08 sec)
```

To repair a table you can use **REPAIR**, this copies as many valid rows from the existing CSV data as possible, and then replaces the existing CSV file with the recovered rows. Any rows beyond the corrupted data are lost.

```
mysql> repair table csvtest;
```

Table	Op	Msg_type	Msg_text
test.csvtest	repair	status	OK

```
1 row in set (0.02 sec)
```

Warning

Note that during repair, only the rows from the CSV file up to the first damaged row are copied to the new table. All other rows from the first damaged row to the end of the table are removed, even valid rows.

13.11.2. CSV Limitations

Important

The **CSV** storage engine does not support indexing.

Partitioning is not supported for tables using the **CSV** storage engine. Beginning with MySQL 5.1.12, it is no longer possible to create partitioned **CSV** tables. (See [Bug#19307](#))

13.12. The **BLACKHOLE** Storage Engine

The **BLACKHOLE** storage engine acts as a “black hole” that accepts data but throws it away and does not store it. Retrievals always return an empty result:

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

To enable the **BLACKHOLE** storage engine if you build MySQL from source, invoke `configure` with the `-with-blackhole-storage-engine` option.

To examine the source for the **BLACKHOLE** engine, look in the `sql` directory of a MySQL source distribution.

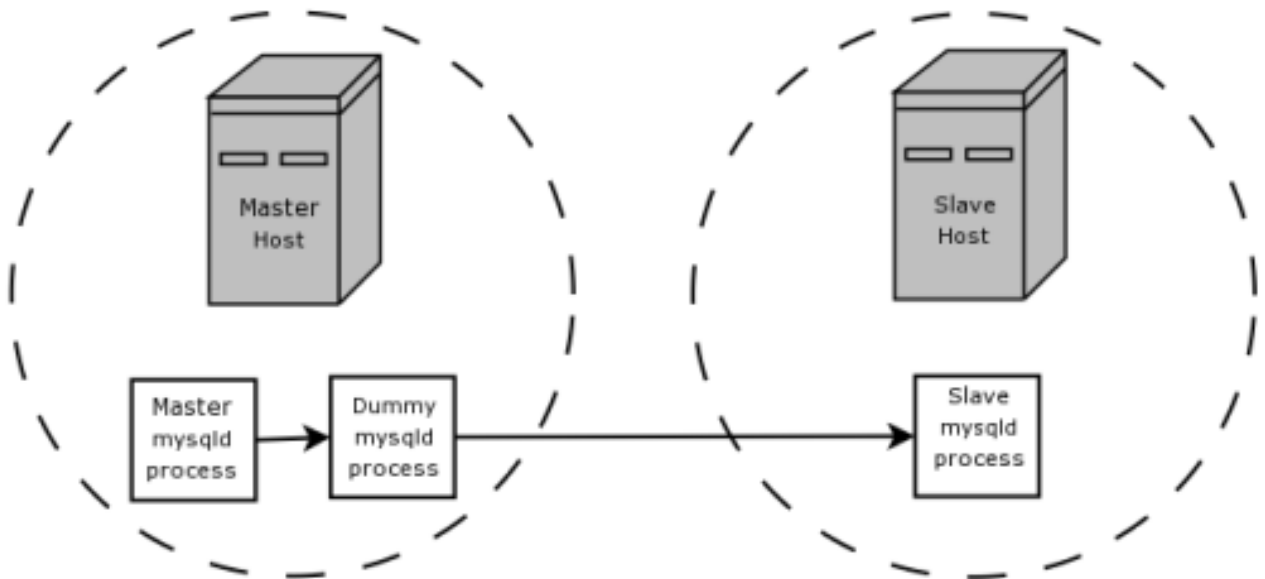
When you create a **BLACKHOLE** table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. There are no other files associated with the table.

The **BLACKHOLE** storage engine supports all kinds of indexes. That is, you can include index declarations in the table definition.

You can check whether the **BLACKHOLE** storage engine is available with this statement:

```
mysql> SHOW VARIABLES LIKE 'have_blackhole_engine';
```

Inserts into a **BLACKHOLE** table do not store any data, but if the binary log is enabled, the SQL statements are logged (and replicated to slave servers). This can be useful as a repeater or filter mechanism. For example, suppose that your application requires slave-side filtering rules, but transferring all binary log data to the slave first results in too much traffic. In such a case, it is possible to set up on the master host a “dummy” slave process whose default storage engine is **BLACKHOLE**, depicted as follows:



The master writes to its binary log. The “dummy” `mysqld` process acts as a slave, applying the desired combination of `replicate-do-*` and `replicate-ignore-*` rules, and writes a new, filtered binary log of its own. (See [Section 15.1.3, “Replication Options and Variables”](#).) This filtered log is provided to the slave.

The dummy process does not actually store any data, so there is little processing overhead incurred by running the additional `mysqld` process on the replication master host. This type of setup can be repeated with additional replication slaves.

INSERT triggers for **BLACKHOLE** tables work as expected. However, because the **BLACKHOLE** table does not actually store any data, **UPDATE** and **DELETE** triggers are not activated: The **FOR EACH ROW** clause in the trigger definition does not apply because there are no rows.

Other possible uses for the **BLACKHOLE** storage engine include:

- Verification of dump file syntax.
- Measurement of the overhead from binary logging, by comparing performance using **BLACKHOLE** with and without binary logging enabled.
- **BLACKHOLE** is essentially a “no-op” storage engine, so it could be used for finding performance bottlenecks not related to the storage engine itself.

The **BLACKHOLE** storage engine does not support **INSERT DELAYED**, **LOCK TABLES**, or **UNLOCK TABLES** statements prior to MySQL 5.1.19.

As of MySQL 5.1.4, the **BLACKHOLE** engine is transaction-aware, in the sense that committed transactions are written to the binary log and rolled-back transactions are not.

Chapter 14. High Availability, Scalability, and DRBD

When using MySQL you may need to ensure the availability or scalability of your MySQL installation. Availability refers to the ability to cope with, and if necessary recover from, failures on the host, including failures of MySQL, the operating system, or the hardware. Scalability refers to the ability to spread the load of your application queries across multiple MySQL servers. As your application and usage increases, you may need to spread the queries for the application across multiple servers to improve response times.

There are a number of solutions available for solving issues of availability and scalability. The two primary solutions supported by MySQL are MySQL Replication and MySQL Cluster. Further options are available using third-party solutions such as DRBD (Distributed Replicated Block Device) and Heartbeat, and more complex scenarios can be solved through a combination of these technologies. These tools work in different ways:

- *MySQL Replication* enables statements and data from one MySQL server instance to be replicated to another MySQL server instance. Without using more complex setups, data can only be replicated from a single master server to any number of slaves. The replication is asynchronous, so the synchronization does not take place in real time, and there is no guarantee that data from the master will have been replicated to the slaves.
 - **Advantages**
 - MySQL Replication is available on all platforms supported by MySQL, and since it isn't operating system-specific it can operate across different platforms.
 - Replication is asynchronous and can be stopped and restarted at any time, making it suitable for replicating over slower links, partial links and even across geographical boundaries.
 - Data can be replicated from one master to any number of slaves, making replication suitable in environments with heavy reads, but light writes (for example, many web applications), by spreading the load across multiple slaves.
 - **Disadvantages**
 - Data can only be written to the master. In advanced configurations, though, you can set up a multiple-master configuration where the data is replicated around a ring configuration.
 - There is no guarantee that data on master and slaves will be consistent at a given point in time. Because replication is asynchronous there may be a small delay between data being written to the master and it being available on the slaves. This can cause problems in applications where a write to the master must be available for a read on the slaves (for example a web application).
 - **Recommended uses**
 - Scale-out solutions that require a large number of reads but fewer writes (for example, web serving).
 - Logging/data analysis of live data. By replicating live data to a slave you can perform queries on the slave without affecting the operation of the master.
 - Online backup (availability), where you need an active copy of the data available. You need to combine this with other tools, such as custom scripts or Heartbeat. However, because of the asynchronous architecture, the data may be incomplete.
 - Offline backup. You can use replication to keep a copy of the data. By replicating the data to a slave, you take the slave down and get a reliable snapshot of the data (without MySQL running), then restart MySQL and replication to catch up. The master (and any other slaves) can be kept running during this period.

For information on setting up and configuring replication, see [Chapter 15, Replication](#).

- *MySQL Cluster* is a synchronous solution that enables multiple MySQL instances to share database information. Unlike replication, data in a cluster can be read from or written to any node within the cluster, and information will be distributed to the other nodes.
 - **Advantages**
 - Offers multiple read and write nodes for data storage.
 - Provides automatic failover between nodes. Only transaction information for the active node being used is lost in the event of a failure.
 - Data on nodes is instantaneously distributed to the other data nodes.

- **Disadvantages**

- Available on a limited range of platforms.
- Nodes within a cluster should be connected via a LAN; geographically separate nodes are not supported. However, you can replicate from one cluster to another using MySQL Replication, although the replication in this case is still asynchronous.

- **Recommended uses**

- Applications that need very high availability, such as telecoms and banking.
- Applications that require an equal or higher number of writes compared to reads.

For information on MySQL Cluster, see [Chapter 16, MySQL Cluster](#).

- *DRBD (Distributed Replicated Block Device)* is a third-party solution from Linbit supported only on Linux. DRBD creates a virtual block device (which is associated with an underlying physical block device) that can be replicated from the primary server to a secondary server. You create a filesystem on the virtual block device, and this information is then replicated, at the block level, to the secondary server.

Because the block device, not the data you are storing on it, is being replicated the validity of the information is more reliable than with data-only replication solutions. DRBD can also ensure data integrity by only returning from a write operation on the primary server when the data has been written to the underlying physical block device on both the primary and secondary servers.

- **Advantages**

- Provides high availability and data integrity across two servers in the event of hardware or system failure.
- Ensures data integrity by enforcing write consistency on the primary and secondary nodes.

- **Disadvantages**

- Only provides a method for duplicating data across the nodes. Secondary nodes cannot use the DRBD device while data is being replicated, and so the MySQL on the secondary node cannot be simultaneously active.
- Cannot provide scalability, since secondary nodes don't have access to the secondary data.

- **Recommended uses**

- High availability situations where concurrent access to the data is not required, but instant access to the active data in the event of a system or hardware failure is required.

For information on configuring DRBD and configuring MySQL for use with a DRBD device, see [Section 14.1, “Using MySQL with DRBD for High Availability”](#).

- *Heartbeat* is a third party software solution for Linux. It is not a data replication or synchronization solution, but a solution for monitoring servers and switching active MySQL servers automatically in the event of failure. Heartbeat needs to be combined with MySQL Replication or DRBD to provide automatic failover.

The information and suitability of the various technologies and different scenarios is summarized in the table below.

Requirements	MySQL Replication	MySQL Replication + Heartbeat	MySQL Heartbeat + DRBD	MySQL Cluster
Availability				
Automated IP failover	No	Yes	Yes	No
Automated database failover	No	No	Yes	Yes
Typical failover time	User/script-dependent	Varies	< 30 seconds	< 3 seconds
Automatic resynchronization of data	No	No	Yes	Yes
Geographic redundancy	Yes	Yes	Yes, when combined with	Yes, when combined with

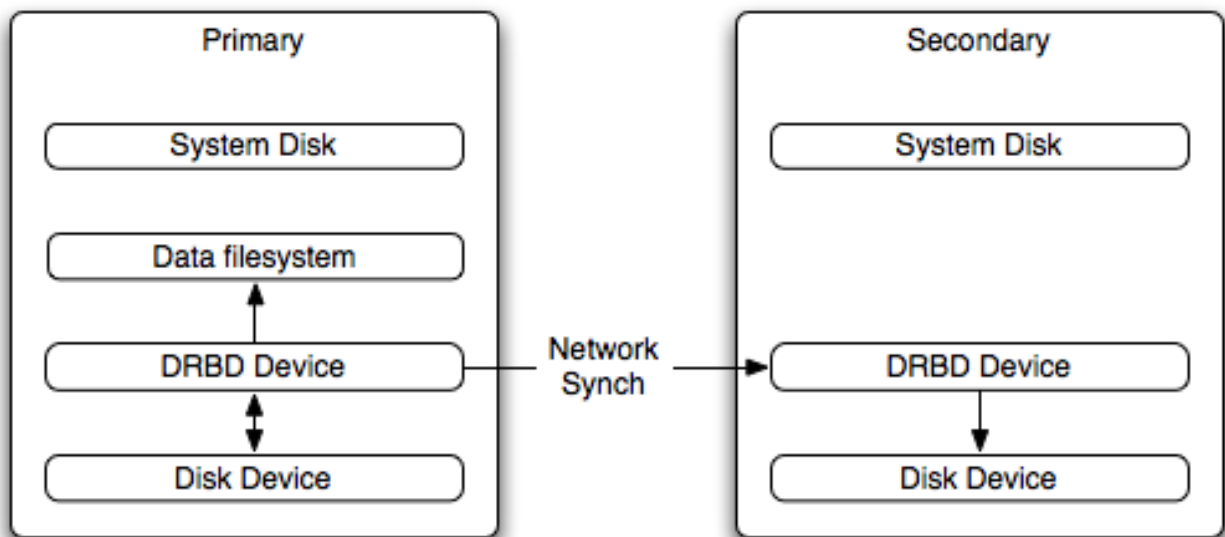
Requirements	MySQL Replication	MySQL Replication + Heartbeat	MySQL Heartbeat + DRBD	MySQL Cluster
support			MySQL Replication	MySQL Replication
Scalability				
Built-in load balancing	No	No	No	Yes
Supports Read-intensive applications	Yes	Yes	Yes, when combined with MySQL Replication	Yes
Supports Write-intensive applications	No	No	Yes	Yes
Maximum number of nodes per group	One master, multiple slaves	One master, multiple slaves	One active (primary), multiple passive (secondary) nodes	255
Maximum number of slaves	Unlimited (reads only)	Unlimited (reads only)	Unlimited (reads only)	Unlimited (reads only)

14.1. Using MySQL with DRBD for High Availability

The Distributed Replicated Block Device (DRBD) is a Linux Kernel module that supports a distributed storage system. You can use DRBD to share block devices between Linux servers and, in turn, share filesystems and data.

DRBD implements a block device which can be used for storage and which is replicated from a primary server to one or more secondary servers. The distributed block device is handled by the DRBD service. Writes to the DRBD block device are distributed among the servers. Each DRBD service writes the information from the DRBD block device to a local physical block device (hard disk).

On the primary, for example, the data writes are written both to the underlying physical block device and distributed to the secondary DRBD services. On the secondary, the writes received through DRBD and written to the local physical block device. On both the primary and the secondary, reads from the DRBD block device are handled by the underlying physical block device. The information is shared between the primary DRBD server and the secondary DRBD server synchronously and at a block level, and this means that DRBD can be used in high-availability solutions where you need failover support.



When used with MySQL, DRBD can be used to ensure availability in the event of a failure. MySQL is configured to store information on the DRBD block device, with one server acting as the primary and a second machine available to operate as an immediate replacement in the event of a failure.

For automatic failover support you can combine DRBD with the Linux Heartbeat project, which will manage the interfaces on the two

servers and automatically configure the secondary (passive) server to replace the primary (active) server in the event of a failure. You can also combine DRBD with MySQL Replication to provide both failover and scalability within your MySQL environment.

In this section we will cover the basics of setting up a DRBD based solution, how to combine this with Linux Heartbeat and at solutions and topologies that make combine DRBD and MySQL Replication.

For information on how to configure DRBD and MySQL, including Heartbeat support, see [Section 14.1.1, “Configuring a MySQL and DRBD Environment”](#).

An FAQ for using DRBD and MySQL is available. See [Section A.14, “MySQL 5.1 FAQ — MySQL, DRBD, and Heartbeat”](#).

Note

Because DRBD is a Linux Kernel module it is currently not supported on platforms other than Linux.

14.1.1. Configuring a MySQL and DRBD Environment

To set up DRBD, MySQL and Heartbeat you need to follow a number of steps that affect the operating system, DRBD and your MySQL installation.

Before starting the installation process, you should be aware of the following information, terms and requirements on using DRBD:

- DRBD is a solution for enabling high-availability, and therefore you need to ensure that the two machines within your DRBD setup are as identically configured as possible so that the secondary machine can act as a direct replacement for the primary machine in the event of system failure.
- DRBD works through two (or more) servers, each called a *node*
- The node that contains the primary data, has read/write access to the data, and in an HA environment is the currently active node is called the *primary*.
- The server to which the data is replicated is designated as *secondary*.
- A collection of nodes that are sharing information are referred to as a *DRBD cluster*.
- For DRBD to operate you must have a block device on which the information can be stored on *each* DRBD node. The *lower level* block device can be a physical disk partition, a partition from a volume group or RAID device or any other block device.

Typically you use a spare partition on which the physical data will be stored . On the primary node, this disk will hold the raw data that you want replicated. On the secondary nodes, the disk will hold the data replicated to the secondary server by the DRBD service. Ideally, the size of the partition on the two DRBD servers should be identical, but this is not necessary as long as there is enough space to hold the data that you want distributed between the two servers.

- For the distribution of data to work, DRBD is used to create a logical block device that uses the lower level block device for the actual storage of information. To store information on the distributed device, a filesystem is created on the DRBD logical block device.
- When used with MySQL, once the filesystem has been created, you move the MySQL data directory (including InnoDB data files and binary logs) to the new filesystem.
- When you set up the secondary DRBD server, you set up the physical block device and the DRBD logical block device that will store the data. The block device data is then copied from the primary to the secondary server.

The overview for the installation and configuration sequence is as follows:

1. First you need to set up your operating system and environment. This includes setting the correct hostname, updating the system and preparing the available packages and software required by DRBD, and configuring a physical block device to be used with the DRBD block device. See [Section 14.1.1.1, “Setting up the Operating System”](#).
2. Installing DRBD requires installing or compiling the DRBD source code and then configuring the DRBD service to set up the block devices that will be shared. See [Section 14.1.1.2, “Installing and Configuring DRBD”](#).
3. Once DRBD has been configured, you must alter the configuration and storage location of the MySQL data. See [Section 14.1.1.3,](#)

“Configuring MySQL for DRBD”.

14.1.1.1. Setting up the Operating System

To set your Linux environment for using DRBD there are a number of system configuration steps that you must follow.

- Make sure that the primary and secondary DRBD servers have the correct hostname, and that the hostnames are unique. You can verify this by using the `uname` command:

```
$ uname -n  
drbd-master
```

If the hostname is not set correctly then edit the appropriate file (usually `/etc/sysconfig/network` or `/etc/conf.d/hostname`) and set the name correctly.

- Each DRBD node must have a unique IP address. Make sure that the IP address information is set correctly within the network configuration and that the hostname and IP address has been set correctly within the `/etc/hosts` file.
- Although you can rely on the DNS or NIS system for host resolving, in the event of a major network failure these services may not be available. If possible, add the IP address and hostname of each DRBD node into the `/etc/hosts` file for each machine. This will ensure that the node information can always be determined even if the DNS/NIS servers are unavailable.
- As a general rule, the faster your network connection the better. Because the block device data is exchanged over the network, everything that will be written to the local disk on the DRBD primary will also be written to the network for distribution to the DRBD secondary.
- You must have a spare disk or disk partition that you can use as the physical storage location for the DRBD data that will be replicated. You do not have to have a complete disk available, a partition on an existing disk is acceptable.

If the disk is unpartitioned, partition the disk using `fdisk`. Do not create a filesystem on the new partition.

Remember that you must have a physical disk available for the storage of the replicated information on each DRBD node. Ideally the partitions that will be used on each node should be of an identical size, although this is not strictly necessary. Do, however, ensure that the physical partition on the DRBD secondary is at least as big as the partitions on the DRBD primary node.

- If possible, upgrade your system to the latest available Linux kernel for your distribution. Once the kernel has been installed, you must reboot to make the kernel active. To use DRBD you will also need to install the relevant kernel development and header files that are required for building kernel modules. Platform specification information for this is available later in this section.

Before you compile or install DRBD, you must make

- Kernel header files
- Kernel source files
- GCC Compiler
- `glib 2`
- `flex`
- `bison`

Additionally, if you want to support secure communication between your DRBD nodes then the following items will be required:

- OpenSSL
- GNU TLS
- GNU Crypt

Here are some operating system specific tips for setting up your installation:

- **Tips for Red Hat (including CentOS and Fedora):**

Use [up2date](#) or [yum](#) to update and install the latest kernel and kernel header files:

```
# up2date kernel-smp-devel kernel-smp
```

Reboot. If you are going to build DRBD from source, then update your system with the required development packages

```
# up2date glib-devel openssl-devel libgcrypt-devel glib2-devel \
pkgconfig ncurses-devel rpm-build rpm-devel redhat-rpm-config gcc \
gcc-c++ bison flex gnutls-devel lm_sensors-devel net-snmp-devel \
python-devel bzip2-devel libselinux-devel perl-DBI
```

If you are going to use the pre-built DRBD RPMs:

```
# up2date gnutls lm_sensors net-snmp ncurses libgcrypt glib2 openssl glib
```

- **Tips for Debian, Ubuntu, Kubuntu:**

Use [apt-get](#) to install the kernel packages

```
# apt-get install linux-headers linux-image-server
```

If you are going to use the pre-built Debian packages for DRBD then you should not need any additional packages.

If you want to build DRBD from source, you will need to use the following command to install the required components:

```
# apt-get install devscripts flex bison build-essential \
dpkg-dev kernel-package debconf-utils dpatch debhelper \
libnet1-dev e2fslibs-dev libglib2.0-dev automake1.9 \
libgnutls-dev libtool libltdl3 libltdl3-dev
```

- **Tips for Gentoo:**

Gentoo is a source based Linux distribution and therefore many of the source files and components that you will need are either already installed or will be installed automatically by [emerge](#).

To install DRBD 0.8.x, you must unmask the [sys-cluster/drbd](#) build by adding the following line to [/etc/portage/package.keywords](#):

```
sys-cluster/drbd ~x86
```

If your kernel does not already have the userspace to kernelspace linker enabled, then you will need to add the following line to your kernel configuration ([/etc/kernels/kernel-config-*](#) file):

```
CONFIG_CONNECTOR=y
```

You can now [emerge](#) DRBD 0.8.x into your Gentoo installation:

```
# emerge drbd
```

14.1.1.2. Installing and Configuring DRBD

To install DRBD you can choose either the pre-built binary installation packages or you can use the source packages and build from source. If you want to build from source you must have installed the source and development packages.

If you are installing using a binary distribution then you must ensure that the kernel version number of the binary package matches your currently active kernel. You can use [uname](#) to find out this information:

```
$ uname -r
2.6.20-gentoo-r6
```

To build from the sources, download the source [tar.gz](#) package, extract the contents and then follow the instructions within the [INSTALL](#) file.

To install DRBD on Gentoo:

```
# emerge drbd
```

Note

Note that this may install an older version of the DRBD software. In this case you must install DRBD from the source tarball by hand.

Once DRBD has been configured and installed you must copy the default DRBD configuration file from [/usr/share/drbd-0.7.23/drbd.conf.bz2](#) to [/etc](#) and then uncompress it using [bunzip2](#).

Once DRBD has been built and installed, you need to edit the [/etc/drbd.conf](#) file and then run a number of commands to build the block device and setup the replication.

Although the steps below are split into those for the primary node and the secondary node, it should be noted that the configuration files for all nodes should be identical, and many of the same steps have to be repeated on each node to enable the DRBD block device.

14.1.1.2.1. Setting up a DRBD Primary Node

To set up a DRBD primary node you need to configure the DRBD service, create the first DRBD block device and then create a filesystem on the device so that you can store files and data.

The DRBD configuration file ([/etc/drbd.conf](#)) defines a number of parameters for your DRBD configuration, including the frequency of updates and block sizes, security information and the definition of the DRBD devices that you want to create.

The key elements to configure are the [on](#) sections which specify the configuration of each node.

To follow the configuration, the sequence below shows only the changes from the default [drbd.conf](#) file. Configurations within the file can be both global, and tied to specific resource (

1. Set the synchronization rate between the two nodes. You should keep this in check compared to the speed of your network connection. Gigabit Ethernet can support up to 125 MB/second, 100Mbps Ethernet slightly less than a tenth of that (12MBps). If you are using a shared network connection, rather than a dedicated, then you should gauge accordingly.

To set the synchronization rate, edit the [rate](#) setting within the [syncer](#) block:

```
syncer {  
    rate 10M;  
}
```

2. Set up some basic authentication. DRBD supports a simple password hash exchange mechanism. This helps to ensure that only those hosts with the same shared secret are able to join the DRBD node group.

```
cram-hmac-alg "sha1";  
shared-secret "shared-string";
```

3. Now you must configure the host information. Remember that you must have the node information for the primary and secondary nodes in the [drbd.conf](#) file on each host. You need to configure the following information for each node:

- [device](#) — the path of the logical block device that will be created by DRBD.
- [disk](#) — the block device that will be used to store the data.
- [address](#) — the IP address and port number of the host that will hold this DRBD device.
- [meta-disk](#) — the location where the metadata about the DRBD device will be stored. You can set this to [internal](#) and DRBD will use the physical block device to store the information, by recording the metadata within the last sections of the disk. The exact size will depend on the size of the logical block device you have created, but it may involve up to 128MB.

A sample configuration for our primary server might look like this:


```
on drbd-master {  
  device /dev/drbd0;  
  disk /dev/hdd1;  
  address 192.168.0.240:8888;  
  meta-disk internal;  
}
```

The **on** configuration block should be repeated for the secondary node (and any further) nodes:

```
on drbd-slave {  
  device /dev/drbd0;  
  disk /dev/hdd1;  
  address 192.168.0.241:8888;  
  meta-disk internal;  
}
```

The IP address of each **on** block must match the IP address of the corresponding host. Do not set this value to the IP address of the corresponding primary or secondary in each case.

4. Before starting the primary node, you should create the metadata for the devices:

```
# drbdmeta create-md all
```

5. You are now ready to start DRBD:

```
# /etc/init.d/drbd start
```

DRBD should now start and initialize, creating the DRBD devices that you have configured.

6. DRBD creates a standard block device - to make it usable, you must create a filesystem on the block device just as you would with any standard disk partition. Before you can create the filesystem, you must mark the new device as the primary device (i.e. where the data will be written and stored), and initialize the device. Because this is a destructive operation, you must specify the command line option to overwrite the raw data:

```
# drbdadm -- --overwrite-data-of-peer primary all
```

If you are using a version of DRBD earlier than 0.8.0, then you need to use a different command-line option:

```
# drbdadm -- --do-what-I-say primary all
```

Now create a filesystem using your chosen filesystem type:

```
# mkfs.ext3 /dev/drbd0
```

7. You can now mount the filesystem and if necessary copy files to the mount point:

```
# mkdir /mnt/drbd  
# mount /dev/drbd0 /mnt/drbd  
# echo "DRBD Device" >/mnt/drbd/samplefile
```

Your primary node is now ready to use. You should now configure your secondary node or nodes.

14.1.1.2.2. Setting up a DRBD Secondary Node

The configuration process for setting up a secondary node is the same as for the primary node, except that you do not have to create the filesystem on the secondary node device, as this information will automatically be transferred from the primary node.

To setup a secondary node:

1. Copy the `/etc/drbd.conf` file from your primary node to your secondary node. It should already contain all the information and configuration that you need, since you had to specify the secondary node IP address and other information for the primary node configuration.

2. Create the DRBD metadata on the underlying disk device:

```
# drbdmeta create-md all
```

3. Start DRBD:

```
# /etc/init.d/drbd start
```

Once DRBD has started, it will start the copy the data from the primary node to the secondary node. Even with an empty filesystem this will take some time, since DRBD is copying the block information from a block device, not simply copying the filesystem data.

You can monitor the progress of the copy between the primary and secondary nodes by viewing the output of [/proc/drbd](#):

```
cat /proc/drbd
version: 8.0.4 (api:86/proto:86)
SVN Revision: 2947 build by root@drbd-master, 2007-07-30 16:43:05
0: cs:SyncSource st:Primary/Secondary ds:UpToDate/Inconsistent C r---
   ns:252284 nr:0 dw:0 dr:257280 al:0 bm:15 lo:0 pe:7 ua:157 ap:0
   [==>.....] sync'ed: 12.3% (1845088/2097152)K
   finish: 0:06:06 speed: 4,972 (4,580) K/sec
   resync: used:1/31 hits:15901 misses:16 starving:0 dirty:0 changed:16
   act_log: used:0/257 hits:0 misses:0 starving:0 dirty:0 changed:0
```

14.1.1.2.3. Monitoring and Managing your DRBD device

Once the primary and secondary machines are configured and synchronized, you can get the status information about your DRBD device is to view the output from [/proc/drbd](#):

```
version: 8.0.4 (api:86/proto:86)
SVN Revision: 2947 build by root@drbd-master, 2007-07-30 16:43:05
0: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r---
   ns:2175704 nr:0 dw:99192 dr:2076641 al:33 bm:128 lo:0 pe:0 ua:0 ap:0
   resync: used:0/31 hits:134841 misses:135 starving:0 dirty:0 changed:135
   act_log: used:0/257 hits:24765 misses:33 starving:0 dirty:0 changed:33
```

The first line provides the version/revision and build information.

The second line starts the detailed status information for an individual resource. The individual field headings are as follows:

- cs — connection state
- st — node state (local/remote)
- ld — local data consistency
- ds — data consistency
- ns — network send
- nr — network receive
- dw — disk write
- dr — disk read
- pe — pending (waiting for ack)
- ua — unack'd (still need to send ack)
- al — access log write count

In the previous example, the information shown indicates that the nodes are connected, the local node is the primary (because it is listed first), and the local and remote data is up to date with each other. The remainder of the information is statistical data about the device, and the data exchanged that kept the information up to date.

For administration, the main command is `drbdadm`. There are a number of commands supported by this tool to control the connectivity and status of the DRBD devices.

The most common commands are those to set the primary/secondary status of the local device. You can manually set this information for a number of reasons, including when you want to check the physical status of the secondary device (since you cannot mount a DRBD device in primary mode), or when you are temporarily moving the responsibility of keeping the data in check to a different machine (for example, during an upgrade or physical move of the normal primary node). You can set state of all local device to be the primary using this command:

```
# drbdadm primary all
```

Or switch the local device to be the secondary using:

```
# drbdadm secondary all
```

To change only a single DRBD resource, specify the resource name instead of `all`.

You can temporarily disconnect the DRBD nodes:

```
# drbdadm disconnect all
```

Reconnect them using `connect`:

```
# drbdadm connect all
```

For other commands and help with `drbdadm` see the DRBD documentation.

14.1.1.2.4. Additional DRBD Configuration Options

Additional options you may want to configure:

- `protocol` — specifies the level of consistency to be used when information is written to the block device. The option is similar in principle to the `innodb_flush_log_at_trx_commit` option within MySQL. Three levels are supported:
 - `A` — data is considered written when the information reaches the TCP send buffer and the local physical disk. There is no guarantee that the data has been written to the remote server or the remote physical disk.
 - `B` — data is considered written when the data has reached the local disk and the remote node's network buffer. The data has reached the remote server, but there is no guarantee it has reached the remote servers physical disk.
 - `C` — data is considered written when the data has reached the local disk and the remote nodes physical disk.

The preferred and recommended protocol is `C`, as it is the only protocol which ensures the consistency of the local and remote physical storage.

- `on-io-error` — specifies what should happen when there is an error with the physical block device. Suitable values include `panic`, which causes the node to leave the DRBD cluster by causing a kernel panic (and which would trigger automatic failover if you were using Heartbeat); and `detach`, which switches DRBD into a diskless mode (i.e. data is not written to the lower level block device).
- `size` — if you do not want to use the entire partition space with your DRBD block device then you can specify the size of the DRBD device to be created. The size specification can include a quantifier. For example, to set the maximum size of the DRBD partition to 1GB you would use:

```
size 1G;
```

With the configuration file suitably configured and ready to use, you now need to populate the lower-level device with the metadata information, and then start the DRBD service.

14.1.1.3. Configuring MySQL for DRBD

Once you have configured DRBD and have an active DRBD device and filesystem, you can configure MySQL to use the chosen device to store the MySQL data.

When performing a new installation of MySQL, you can either select to install MySQL entirely onto the DRBD device, or just configure the data directory to be located on the new filesystem.

In either case, the files and installation must take place on the primary node, because that is the only DRBD node on which you can mount the DRBD device filesystem as read/write.

You should store the following files and information on your DRBD device:

- MySQL data files, including the binary log, and InnoDB data files.
- MySQL configuration file (`my.cnf`).

To setup MySQL to use your new DRBD device and filesystem:

1. If you are migrating an existing MySQL installation, stop MySQL:

```
$ mysqladmin shutdown
```

2. Copy the `my.cnf` onto the DRBD device. If you are not already using a configuration file, copy one of the sample configuration files from the MySQL distribution.

```
# mkdir /mnt/drbd/mysql
# cp /etc/my.cnf /mnt/drbd/mysql
```

3. Copy your MySQL data directory to the DRBD device and mounted filesystem.

```
# cp -R /var/lib/mysql /drbd/mysql/data
```

4. Edit the configuration file to reflect the change of directory by setting the value of the `datadir` option. If you have not already enabled the binary log, also set the value of the `log-bin` option.

```
datadir = /drbd/mysql/data
log-bin = mysql-bin
```

5. Create a symbolic link from `/etc/my.cnf` to the new configuration file on the DRBD device filesystem.

```
# ln -s /drbd/mysql/my.cnf /etc/my.cnf
```

6. Now start MySQL and check that the data that you copied to the DRBD device filesystem is present.

```
# /etc/init.d/mysql start
```

Your MySQL data should now be located on the filesystem running on your DRBD device. The data will be physically stored on the underlying device that you configured for the DRBD device. Meanwhile, the content of your MySQL databases will be copied to the secondary DRBD node.

Note that you cannot access the information on your secondary node, as a DRBD device working in secondary mode is not available for use.

Chapter 15. Replication

Replication enables data from one MySQL database server (called the master) to be replicated to one or more MySQL database servers (slaves). Replication is asynchronous - your replication slaves do not need to be connected permanently to receive updates from the master, which means that updates can occur over long-distance connections and even temporary solutions such as a dial-up service. Depending on the configuration, you can replicate all databases, selected databases and even selected tables within a database.

The target uses for replication in MySQL include:

- Scale-out solutions - spreading the load among multiple slaves to improve performance. In this environment, all writes and updates must take place on the master server. Reads, however, may take place on one or more slaves. This model can improve the performance of writes (since the master is dedicated to updates), while dramatically increasing read speed across an increasing number of slaves.
- Data security - because data is replicated to the slave, and the slave can pause the replication process, it is possible to run backup services on the slave without corrupting the corresponding master data.
- Analytics - live data can be created on the master, while the analysis of the information can take place on the slave without affecting the performance of the master.
- Long-distance data distribution - if a branch office would like to work with a copy of your main data, you can use replication to create a local copy of the data for their use without requiring permanent access to the master.

Replication in MySQL features support for one-way, asynchronous replication, in which one server acts as the master, while one or more other servers act as slaves. This is in contrast to the *synchronous* replication which is a characteristic of MySQL Cluster (see [Chapter 16, MySQL Cluster](#)).

There are a number of solutions available for setting up replication between two servers, but the best method to use depends on the presence of data and the engine types you are using. For more information on the available options, see [Section 15.1.1, “How to Set Up Replication”](#).

There are two core types of replication format, Statement Based Replication (SBR), which replicates entire SQL statements, and Row Based Replication (RBR), which replicates only the changed rows. You may also use a third variety, Mixed Based Replication (MBR), which is the default mode within MySQL 5.1.14 and later. For more information on the different replication formats, see [Section 15.1.2, “Replication Formats”](#).

Replication is controlled through a number of different options and variables. These control the core operation of the replication, timeouts and the databases and filters that can be applied on databases and tables. For more information on the available options, see [Section 15.1.3, “Replication Options and Variables”](#).

You can use replication to solve a number of different problems, including problems with performance, supporting the backup of different databases and for use as part of a larger solution to alleviate system failures. For information on how to address these issues, see [Section 15.2, “Replication Solutions”](#).

For notes and tips on how different data types and statements are treated during replication, including details of replication features, version compatibility, upgrades, and problems and their resolution, including an FAQ, see [Section 15.3, “Replication Notes and Tips”](#).

Detailed information on the implementation of replication, how replication works, the process and contents of the binary log, background threads and the rules used to decide how statements are recorded and replication, see [Section 15.4, “Replication Implementation”](#).

MySQL Enterprise

The MySQL Enterprise Monitor provides numerous advisors that provide immediate feedback about replication-related problems. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

15.1. Replication Configuration

Replication between servers in MySQL works through the use of the binary logging mechanism. The MySQL instance operating as the master (the source of the database changes) writes updates and changes to the database to the binary log. The information in the binary log is stored in different logging formats according to the database changes being recorded. Slaves are configured to read the binary log from the master and to execute the events in the binary log on the slave's local database.

The Master is dumb in this scenario. Once binary logging has been enabled, all statements are recorded in the binary log. Each slave will receive a copy of the entire contents of the binary log. It is the responsibility of the slave to decide which statements in the binary log should be executed; you cannot configure the master to log only certain events. If you do not specify otherwise, all events in the master binary log are executed on the slave. If required, you can configure the slave to only process events that apply to particular databases or tables.

Slaves keep a record of the binary log file and position within the log file that they have read and processed from the master. This means that multiple slaves can be connected to the master and executing different parts of the same binary log. Because the slaves control this process, individual slaves can be connected and disconnected from the server without affecting the master's operation. Also, because each slave remembers the position within the binary log, it is possible for slaves to be disconnected, reconnect and then 'catch up' by continuing from the recorded position.

Both the master and each slave must be configured with a unique id (using the `server-id` option). In addition, the slave must be configured with information about the master host name, log file name and position within that file. These details can be controlled from within a MySQL session using the `CHANGE MASTER` statement. The details are stored within the `master.info` file.

In this section the setup and configuration required for a replication environment is described, including step-by-step instructions for creating a new replication environment. The major components of this section are:

- For a guide to setting up two or more servers for replication see [Section 15.1.1, “How to Set Up Replication”](#). This section deals with the setup of the systems and provides methods for copying data between the master and slaves.
- Events in the binary log are recorded using a number of formats. These are referred to as statement based replication (SBR) or row based replication (RBR). A third type, mixed-format replication (MIXED), uses SBR or RBR replication automatically to take advantage of the benefits of both SBR and RBR formats when appropriate. The different formats are discussed in [Section 15.1.2, “Replication Formats”](#).
- Detailed information on the different configuration options and variables that apply to replication is provided in [Section 15.1.3, “Replication Options and Variables”](#).
- Once started, the replication process should require little administration or monitoring. However, for advice on common tasks that you may want to executed, see [Section 15.1.4, “Common Replication Administration Tasks”](#).

15.1.1. How to Set Up Replication

This section describes how to set up complete replication of a MySQL server. There are a number of different methods for setting up replication, and the exact method that you use will depend on how you are setting up replication, and whether you already have data within your master database.

There are some generic tasks which may be required for all replication setups:

- You may want to create a separate user that will be used by your slaves to authenticate with the master to read the binary log for replication. The step is optional. See [Section 15.1.1.1, “Creating a User For Replication”](#).
- You must configure the master to support the binary log and configure a unique ID. See [Section 15.1.1.2, “Setting the Replication Master Configuration”](#).
- You must configure a unique ID for each slave that you want to connect to the Master. See [Section 15.1.1.3, “Setting the Replication Slave Configuration”](#).
- Before starting a data snapshot or the replication process, you should record the position of the binary log on the master. You will need this information when configuring the slave so that the slave knows where within the binary log to start executing events. See [Section 15.1.1.4, “Obtaining the Master Replication Information”](#).
- If you already have data on your Master and you want to synchronize your slave with this base data, then you will need to create a data snapshot of your database. You can create a snapshot using `mysqldump` (see [Section 15.1.1.5, “Creating a Data Snapshot using mysqldump”](#)) or by copying the data files directly (see [Section 15.1.1.6, “Creating a Data Snapshot Using Raw Data Files”](#)).
- You will need to configure the slave with the Master settings, such as the hostname, login credentials and binary log name and positions. See [Section 15.1.1.10, “Setting the Master Configuration on the Slave”](#).

Once you have configured the basic options, you will need to follow the instructions for your replication setup. A number of alternatives are provided:

- If you are setting up a new MySQL master and one or more slaves, then you need only setup up the configuration, as you have no data to exchange. For guidance on setting up replication in this situation, see [Section 15.1.1.7, “Setting up Replication with new Master and Slaves”](#).
- If you are already running a MySQL server, and therefore already have data that will need to be transferred to your slaves before replication starts, have not previously configured the binary log and are able to shutdown your MySQL server for a short period during the process, see [Section 15.1.1.8, “Setting up replication with existing data”](#).
- If you are setting up additional slaves to an existing replication environment then you can setup the slaves without affecting the master. See [Section 15.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”](#).

If you want to administer a MySQL replication setup, we suggest that you read this entire chapter through and try all statements mentioned in [Section 12.6.1, “SQL Statements for Controlling Master Servers”](#), and [Section 12.6.2, “SQL Statements for Controlling Slave Servers”](#). You should also familiarize yourself with the replication startup options described in [Section 15.1.3, “Replication Options and Variables”](#).

Note

Note that certain steps within the setup process require the [SUPER](#) privilege. If you do not have this privilege then enabling replication may not be possible.

15.1.1.1. Creating a User For Replication

Each Slave must connect to the Master using a standard username and password. The user that you use for this operation can be any user, providing they have been granted the [REPLICATION SLAVE](#) privilege.

You do not need to create a specific user for replication. However, you should be aware that the username and password will be stored in plain text within the `master.info` file. Therefore you may want to create a user that only has privileges for the replication process.

To create a user or grant an existing user the privileges required for replication use the [GRANT](#) statement. If you create a user solely for the purposes of replication then that user only needs the [REPLICATION SLAVE](#) privilege. For example, to create a user, `repl`, that allows all hosts within the domain `mydomain.com` to connect for replication:

```
mysql> GRANT REPLICATION SLAVE ON *.*  
-> TO 'repl'@'%mydomain.com' IDENTIFIED BY 'slavepass';
```

See [Section 12.5.1.3, “GRANT Syntax”](#), for more information on the [GRANT](#) statement.

You may wish to create a different user for each slave, or use the same user for each slave that needs to connect. As long as each user that you want to use for the replication process has the [REPLICATION SLAVE](#) privilege you can create as many users as you require.

15.1.1.2. Setting the Replication Master Configuration

For replication to work you *must* enable binary logging on the master. If binary logging is not enabled, replication will not be possible as it is the binary log that is used to exchange data between the master and slaves.

Each server within a replication group must have a unique [server-id](#). The server-id is used to identify individual servers within the group, and must be positive integer between 1 and $(2^{32}-1)$. How you organize and select the numbers is entirely up to you.

To configure both these options you will need to shutdown your MySQL server and edit the configuration of the `my.cnf` or `my.ini` file.

You will need to add the following options to the configuration file within the `[mysqld]` section. If these options already exist, but are commented out, uncomment the options and alter them according to your needs. For example, to enable binary logging, using a log filename prefix of `mysql-bin`, and setting a server ID of 1:

```
[mysqld]  
log-bin=mysql-bin  
server-id=1
```

Note

For the greatest possible durability and consistency in a replication setup using **InnoDB** with transactions, you should use `innodb_flush_log_at_trx_commit=1` and `sync_binlog=1` in the master `my.cnf` file.

Note

Ensure that the `skip-networking` option has not been enabled on your replication master. If networking has been disabled, then your slave will not be able to communicate with the master and replication will fail.

15.1.1.3. Setting the Replication Slave Configuration

The only option you must configure on the slave is to set the unique server ID. If this option is not already set, or the current value conflicts with the value that you have chosen for the master server, then you should shutdown your slave server, and edit the configuration to specify the server id. For example:

```
[mysqld]
server-id=2
```

If you are setting up multiple slaves, each one must have a unique `server-id` value that differs from that of the master and from each of the other slaves. Think of `server-id` values as something similar to IP addresses: These IDs uniquely identify each server instance in the community of replication partners.

If you do not specify a `server-id` value, it is set to 1 if you have not defined `master-host`; otherwise it is set to 2. Note that in the case of `server-id` omission, a master refuses connections from all slaves, and a slave refuses to connect to a master. Thus, omitting `server-id` is good only for backup with a binary log.

You do not have to enable binary logging on the slave for replication to be enabled. However, if you enable binary logging on the slave then you can use the binary log for data backups and crash recovery on the slave, and also use the slave as part of a more complex replication topology.

15.1.1.4. Obtaining the Master Replication Information

To configure replication on the slave you must determine the master's current point within the master binary log. You will need this information so that when the slave starts the replication process, it is able to start processing events from the binary log at the correct point.

If you have existing data on your master that you want to synchronize on your slaves before starting the replication process, then you must stop processing statements on the master, obtain the current position, and then dump the data, before allowing the master to continue executing statements. If you do not stop the execution of statements then the data dump, the master status information that you use will not match and you will end up with inconsistent or corrupted databases on the slaves.

To get the master status information, follow these steps:

1. Start the command line client and flush all tables and block write statements by executing the `FLUSH TABLES WITH READ LOCK` statement:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

For **InnoDB** tables, note that `FLUSH TABLES WITH READ LOCK` also blocks `COMMIT` operations.

Warning

Leave the client from which you issued the `FLUSH TABLES` statement running so that the read lock remains in effect. If you exit the client, the lock is released.

2. Use the `SHOW MASTER STATUS` statement to determine the current binary log name and offset on the master:

```
mysql > SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.003  | 73       | test         | manual,mysql      |
+-----+-----+-----+-----+
```


The **File** column shows the name of the log and **Position** shows the offset within the file. In this example, the binary log file is `mysql-bin.003` and the offset is 73. Record these values. You need them later when you are setting up the slave. They represent the replication coordinates at which the slave should begin processing new updates from the master.

If the master has been running previously without binary logging enabled, the log name and position values displayed by `SHOW MASTER STATUS` or `mysqldump --master-data` will be empty. In that case, the values that you need to use later when specifying the slave's log file and position are the empty string (`' '`) and 4.

You now have the information you need to enable the slave to start reading from the binary log in the correct place to start replication.

If you have existing data that needs to be synchronised with the slave before you start replication, leave the client running so that the lock remains in place and then proceed to [Section 15.1.1.5, “Creating a Data Snapshot using mysqldump”](#), or [Section 15.1.1.6, “Creating a Data Snapshot Using Raw Data Files”](#).

If you are setting up a brand new master and slave replication group, then you can exit the client and release the locks.

15.1.1.5. Creating a Data Snapshot using `mysqldump`

One way to create a snapshot of the data in an existing master database is to use the `mysqldump` tool. Once the data dump has been completed, you then import this data into the slave before starting the replication process.

To obtain a snapshot of the data using `mysqldump`:

- If you haven't already locked the tables on the server to prevent queries that update data from executing:

Start the command line client and flush all tables and block write statements by executing the `FLUSH TABLES WITH READ LOCK` statement:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

Remember to use `SHOW MASTER STATUS` and record the binary log details for use when starting up the slave. The point in time of your snapshot and the binary log position must match. See [Section 15.1.1.4, “Obtaining the Master Replication Information”](#).

- In another session, use `mysqldump` to create a dump either of all the databases you want to replicate, or by selecting specific databases individually. For example:

```
shell> mysqldump --all-databases --lock-all-tables >dbdump.db
```

- An alternative to using a bare dump, is to use the `--master-data` option, which will automatically append the `CHANGE MASTER` statement required on the slave to start the replication process.

```
shell> mysqldump --all-databases --master-data >dbdump.db
```

When choosing databases to include in the dump, remember that you will need to filter out databases on each slave that you do not want to include in the replication process.

You will need either to copy the dump file to the slave, or to use the file from the master when connecting remotely to the slave to import the data.

15.1.1.6. Creating a Data Snapshot Using Raw Data Files

If your database is particularly large then copying the raw data files may be more efficient than using `mysqldump` and importing the file on each slave.

However, using this method with tables in storage engines with complex cache and/or logging algorithms may not give you a perfect 'in time' snapshot as cache information and logging updates may not have been applied, even if you have acquired a global read lock. How the storage engine responds to this will depend on the crash recovery abilities.

For example, when you have acquired a global read lock, you can start a filesystem snapshot of your **InnoDB** tables. Internally (inside the **InnoDB** storage engine) the snapshot won't be consistent (because the **InnoDB** caches are not flushed), but this is not a cause for

concern, because **InnoDB** resolves this at startup and delivers a consistent result. This means that **InnoDB** can perform crash recovery when started on this snapshot, without corruption. However, there is no way to stop the MySQL server while insuring a consistent snapshot of your **InnoDB** tables.

To create your raw data snapshot you can use standard copy tools such as **cp** or **copy**, a remote copy tool such as **scp** or **rsync** an archiving tool such as **zip** or **tar**, or a file system snapshot tool such as **dump**, providing that your MySQL data files exist on a single filesystem. If you are only replicating certain databases then make sure you only copy those files that related to those tables. For **InnoDB**, all tables in all databases are stored in a single file unless you have the **innodb_file_per_table** option enabled.

You may want to specifically exclude the following files from your archive:

- Files relating to the **mysql** database.
- The **master.info** file.
- The master's binary log files.
- Any relay log files.

To get the most consistent results with a raw data snapshot you should shut down the server during the process, as below:

1. Acquire a read lock and get the master status. See [Section 15.1.1.4, “Obtaining the Master Replication Information”](#).
2. In a separate session, shutdown the MySQL server:

```
shell> mysqladmin shutdown
```

3. Take a copy of the MySQL data files. Examples are shown below for common solutions - you need to choose only one of these solutions:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

4. Startup the MySQL instance on the master.

To get a snapshot of the system from a master without shutting down the database:

1. Acquire a read lock and get the master status. See [Section 15.1.1.4, “Obtaining the Master Replication Information”](#).
2. Take a copy of the MySQL data files. Examples are shown below for common solutions - you need to choose only one of these solutions:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

If you are using **InnoDB** tables, ideally you should use the **InnoDB Hot Backup** tool, which takes a consistent snapshot without acquiring any locks on the master server, and records the log name and offset corresponding to the snapshot to be later used on the slave. **Hot Backup** is an additional non-free (commercial) tool that is not included in the standard MySQL distribution. See the **InnoDB Hot Backup** home page at <http://www.innodb.com/manual.php> for detailed information.

3. In the client where you acquired read lock, free the lock:

```
mysql> UNLOCK TABLES;
```

Once you have created the archive or copy of the database, you will need to copy the files to each slave before starting the slave replication process.

15.1.1.7. Setting up Replication with new Master and Slaves

Setting up replication with a new Master and Slaves (i.e. with no existing data) is the easiest and most straightforward method for setting up replication.

You can also use this method if you are setting up new servers and have an existing dump of the databases that you want to load into your replication configuration. By loading the data onto a new master, the data will be automatically replicated to the slaves.

To set up replication between a new master and slave:

1. Configure the MySQL master with the necessary configuration properties. See [Section 15.1.1.2, “Setting the Replication Master Configuration”](#).
2. Start up the MySQL master.
3. Setup a user, see [Section 15.1.1.1, “Creating a User For Replication”](#).
4. Obtain the master status information. See [Section 15.1.1.4, “Obtaining the Master Replication Information”](#).
5. Free the read lock:

```
mysql> UNLOCK TABLES;
```

6. On the slave, edit the MySQL configuration. See [Section 15.1.1.3, “Setting the Replication Slave Configuration”](#).
7. Start up the MySQL slave.
8. Execute the `CHANGE MASTER` command to set the master replication server configuration.

Because there is no data to load or exchange on a new server configuration you do not need to copy or import any information.

If you are setting up a new replication environment using the data from an existing database server, you will now need to run the dump file on the master. The database updates will automatically be propagated to the slaves:

```
shell> mysql -h master < fulldb.dump
```

15.1.1.8. Setting up replication with existing data

When setting up replication with existing data, you will need to decide how best to get the data from the master to the slave before starting the replication service.

The basic process for setting up replication with existing data is as follows:

1. If you have not already configured the `server-id` and binary logging, you will need to shutdown your master to configure these options. See [Section 15.1.1.2, “Setting the Replication Master Configuration”](#).

If you have to shut down your master database, then this is a good opportunity to take a snapshot of the database. You should obtain the master status (see [Section 15.1.1.4, “Obtaining the Master Replication Information”](#)) before taking the database down, updating the configuration and taking a snapshot. For information on how to create a snapshot using raw data files, see [Section 15.1.1.6, “Creating a Data Snapshot Using Raw Data Files”](#).

2. If your server is already correctly configured, obtain the master status (see [Section 15.1.1.4, “Obtaining the Master Replication Information”](#)) and then use `mysqldump` to take a snapshot (see [Section 15.1.1.5, “Creating a Data Snapshot using mysqldump”](#)) or take a raw snapshot of the live database using the guide in [Section 15.1.1.6, “Creating a Data Snapshot Using Raw Data Files”](#).
3. With the MySQL master running, create a user to be used by the slave when connecting to the master during replication. See [Section 15.1.1.1, “Creating a User For Replication”](#).
4. Update the configuration of the slave, see [Section 15.1.1.3, “Setting the Replication Slave Configuration”](#).
5. The next step depends on how you created the snapshot of data on the master.

If you used `mysqldump`:

- a. Startup the slave, skipping replication by using the `--skip-slave` option.
- b. Import the dump file:

```
shell> mysql < fulldb.dump
```

If you created a snapshot using the raw data files:

- a. Extract the data files into your slave data directory. For example:

```
shell> tar xvf dbdump.tar
```

You may need to set permissions and ownership on the files to match the configuration of your slave.

- b. Startup the slave, skipping replication by using the `--skip-slave` option.
6. Configure the slave with the master status information. This will tell the slave the binary log file and position within the file where replication needs to start, and configure the login credentials and hostname of the master. For more information on the statement required, see [Section 15.1.1.10, “Setting the Master Configuration on the Slave”](#).
7. Start the slave threads:

```
mysql> START SLAVE;
```

After you have performed this procedure, the slave should connect to the master and catch up on any updates that have occurred since the snapshot was taken.

If you have forgotten to set the `server-id` option for the master, slaves cannot connect to it.

If you have forgotten to set the `server-id` option for the slave, you get the following error in the slave's error log:

```
Warning: You should set server-id to a non-0 value if master_host
is set; we will force server id to 2, but this MySQL server will
not act as a slave.
```

You also find error messages in the slave's error log if it is not able to replicate for any other reason.

Once a slave is replicating, you can find in its data directory one file named `master.info` and another named `relay-log.info`. The slave uses these two files to keep track of how much of the master's binary log it has processed. Do *not* remove or edit these files unless you know exactly what you are doing and fully understand the implications. Even in that case, it is preferred that you use the `CHANGE MASTER TO` statement to change replication parameters. The slave will use the values specified in the statement to update the status files automatically.

Note

The content of `master.info` overrides some of the server options specified on the command line or in `my.cnf`. See [Section 15.1.3, “Replication Options and Variables”](#), for more details.

Once you have a snapshot of the master, you can use it to set up other slaves by following the slave portion of the procedure just described. You do not need to take another snapshot of the master; you can use the same one for each slave.

15.1.1.9. Introducing Additional Slaves to an Existing Replication Environment

If you want to add another slave to the existing replication configuration then you can do so without stopping the master. Instead, you duplicate the settings on the slaves.

To duplicate the slave:

1. Shutdown the existing slave (slavea):

```
shell> mysqladmin shutdown
```

2. Copy the data directory from the existing slave to the new slave. You can do this by creating an archive using [tar](#) or [WinZip](#), or by performing a direct copy using a tool such as [cp](#) or [rsync](#). Ensure you also copy the log files and relay log files.
3. Copy the `master.info` and `relay.info` files from the existing slave to the new slave. These files hold the current log positions.
4. Start the existing slave.
5. On the new slave, edit the configuration and give the new slave a new unique `server-id`.
6. Start the new slave; the `master.info` file options will be used to start the replication process.

15.1.1.10. Setting the Master Configuration on the Slave

To setup the slave to communicate with the master for replication, you must tell the slave the necessary connection information. To do this, execute the following statement on the slave, replacing the option values with the actual values relevant to your system:

```
mysql> CHANGE MASTER TO
-> MASTER_HOST='master_host_name',
-> MASTER_USER='replication_user_name',
-> MASTER_PASSWORD='replication_password',
-> MASTER_LOG_FILE='recorded_log_file_name',
-> MASTER_LOG_POS=recorded_log_position;
```

The following table shows the maximum allowable length for the string-valued options:

MASTER_HOST	60
MASTER_USER	16
MASTER_PASSWORD	32
MASTER_LOG_FILE	255

15.1.2. Replication Formats

Replication works because events written to the binary log are read from the master and then processed on the slave. The events are recorded within the binary log in different formats according to the type of event being recorded. The different replication formats used correspond to the binary logging format used when the events were recorded in the master's binary log. The correlation between binary logging formats and the terms used during replication are:

- Replication capabilities in MySQL originally were based on propagation of SQL statements from master to slave. This is called *statement-based replication* (SBR) and this correlates to the standard statement-based binary logging format. Binary logging and replication in MySQL 5.1.4 and earlier, and all previous versions of MySQL, used this format.
- Row-based binary logging logs the physical changes to individual table rows. In replication terms this is *row-based replication* (RBR), the master writes events to the binary log that indicate how individual table rows are affected. Support for RBR was added in MySQL 5.1.5.
- As of MySQL 5.1.8, the binary logging format can be altered on the fly according to the event being logged. With mixed-based logging (and the associated *mixed-based replication* (MBR)), statement-based logging is used by default, but automatically switches to row-based logging in particular cases as described below. See [Section 5.11.4.3, “Mixed Binary Logging \(MBL\) Format”](#).

Starting with MySQL 5.1.12, mixed-based replication (i.e. mixed-based logging) is the default format for all replication environment unless you specify otherwise.

Starting with MySQL 5.1.20, the binary logging format used is partially determined by the storage engine being used and the statement being executed. For more information on mixed-based logging and the rules governing the support of different logging formats, see [Section 5.11.4.3, “Mixed Binary Logging \(MBL\) Format”](#).

There are some known limitations and issues between the different replication formats. For a comparison that shows the advantages and disadvantages of statement-based and row-based replication, see [Section 15.1.2.1, “Comparison of Statement-Based Versus Row-Based Replication”](#).

MySQL Cluster Replication makes use of row-based replication. The NDB storage engine is incompatible with statement-based replication and NDB sets row-based logging format automatically. For more information, see [Section 16.10, “MySQL Cluster Replication”](#).

With MySQL's classic statement-based replication, there may be issues with replicating stored routines or triggers. You can avoid these issues by using MySQL's row-based replication instead. For a detailed list of issues, see [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

If you build MySQL from source, row-based replication is available by default unless you invoke `configure` with the `-without-row-based-replication` option.

For MySQL 5.1.20 and later (and MySQL 5.0.46 for backwards compatibility), the following session variables are written to the binary log and honoured by the replication slave when parsing the binary log:

- `SQL_MODE`
- `FOREIGN_KEY_CHECKS`
- `UNIQUE_CHECKS`
- `CHARACTER_SET_CLIENT`
- `COLLATION_CONNECTION`
- `COLLATION_DATABASE`
- `COLLATION_SERVER`
- `SQL_AUTO_IS_NULL`

15.1.2.1. Comparison of Statement-Based Versus Row-Based Replication

Each binary logging format has advantages and disadvantages. For most users, the mixed-based replication format should be fine and should provide the best combination of data integrity and performance. If, however, you want to take advantage of the differences in the replication format when performing specific updates or large data inserts, then the information in this section summarizes the advantages and disadvantages of the row and statement based formats.

Advantages of statement-based replication:

- Proven technology that has existed in MySQL since 3.23.
- Smaller log files. When updates or deletes affect many rows, *much* smaller log files. Smaller log files require less storage space and are faster to back up.
- Log files contain all statements that made any changes, so they can be used to audit the database.
- Log files can be used for point-in-time recovery, not just for replication purposes. See [Section 5.9.3, “Point-in-Time Recovery”](#).
- You can use a slave with a higher version than that used on the master, even when there is a different row structure in the table. This can be useful if you are unable to upgrade the master but want to take advantage of features in a recent slave version, perhaps for testing and evaluation purposes.

Disadvantages of statement-based replication:

- Not all `UPDATE` statements can be replicated: Any non-deterministic behavior (for example, when using random functions in an SQL statement) is hard to replicate when using statement-based replication. For statements that use a non-deterministic user-defined function (UDF), it is not possible to replicate the result using statement-based replication, whereas row-based replication will just replicate the value returned by the UDF.
- Statements cannot be replicated properly if they use a UDF that is non-deterministic (its value depends on other factors than the given parameters).

- Statements that use one of the following functions cannot be replicated properly:

- `LOAD_FILE()`
- `UUID()`
- `USER()`
- `FOUND_ROWS()`
- `SYSDATE()` (unless the server is started with the `--sysdate-is-now` option)

All other functions are replicated correctly (including `RAND()`, `NOW()`, `LOAD DATA INFILE`, and so forth).

- `INSERT ... SELECT` requires a greater number of row-level locks than with row-based replication.
- `UPDATE` statements that require a table scan (because no index is used in the `WHERE` clause) must lock a greater number of rows than with row-based replication.
- For `InnoDB`: An `INSERT` statement that uses `AUTO_INCREMENT` blocks other non-conflicting `INSERT` statements.
- For complex queries, the statement must be evaluated and executed on the slave before the rows are updated or inserted. With row-based replication, the slave only has to run the statement to apply the differences, not the full query.
- Stored functions (not stored procedures) will execute with the same `NOW()` value as the calling statement. (This may be regarded both as a bad thing and a good thing.)
- Deterministic UDFs must be applied on the slaves.
- If there is an error in evaluation on the slave, particularly when executing complex queries, then using statement based replication may slowly increase the margin of error across the affected rows over time.
- Tables have to be (almost) identical on master and slave.

Advantages of row-based replication:

- Everything can be replicated. This is the safest form of replication.

For MySQL versions earlier than 5.1.14, DDL (data definition language) statements such as `CREATE TABLE` are replicated using statement-based replication, while DML (data manipulation language) statements, as well as `GRANT` and `REVOKE` statements, are replicated using row-based-replication.

For MySQL 5.1.14 and later, the `mysql` database is not replicated. The `mysql` database is instead seen as a node specific database. Row-based replication is not supported on this table. Instead, statements that would normally update this information (including `GRANT`, `REVOKE` and the manipulation of triggers, stored routines/procedures and views are all replicated to slaves using Statement based replication.

For statements like `CREATE ... SELECT`, a `CREATE` statement is generated from the table definition and replicated statement-based, while the row insertions are replicated row-based.

- The technology is the same as most other database management systems; knowledge about other systems transfers to MySQL.
- In many cases, it is faster to apply data on the slave for tables that have primary keys.
- Fewer locks needed (and thus higher concurrency) on the master for the following types of statements:
 - `INSERT ... SELECT`
 - `INSERT` statements with `AUTO_INCREMENT`
 - `UPDATE` or `DELETE` statements with `WHERE` clauses that don't use keys or don't change most of the examined rows.
- Fewer locks on the slave for any `INSERT`, `UPDATE`, or `DELETE` statement.
- It's possible to add multiple threads to apply data on the slave in the future (works better on SMP machines).

Disadvantages of row-based replication:

- Larger log files (much larger in some cases).
- Binary log will contain data for large statements that were rolled back.
- When using row-based replication to replicate a statement (for example, an [UPDATE](#) or [DELETE](#) statement), each changed row must be written to the binary log. In contrast, when using statement-based replication, only the statement is written to the binary log. If the statement changes many rows, row-based replication may write significantly more data to the binary log. In these cases the binary log will be locked for a longer time to write the data, which may cause concurrency problems.
- Deterministic UDFs that generate large [BLOB](#) values will be notably slower to replicate.
- You cannot examine the logs to see what statements were executed.
- You cannot see on the slave what statements were received from the master and executed.
- When making a bulk operation that includes non-transactional storage engines, changes are applied as the statement executes. With row-based replication logging, this means that the binary log is written while the statement is running. On the master, this doesn't provide any problems with concurrency, because tables are locked until the bulk operation terminates. On the slave server, however, tables aren't locked while the slave applies changes, because it doesn't know that those changes are part of a bulk operation.

In that scenario, if you retrieve data from a table on the master (for example, [SELECT * FROM table_name](#)), the server will wait for the bulk operation to complete before executing the [SELECT](#) statement, because the table is read-locked. On the slave, the server won't wait (because there is no lock). This means that until the “bulk operation” on the slave has completed you will get different results for the same [SELECT](#) query on the master and on the slave.

This behavior will eventually change, but until it does, you should probably use statement-based replication in a scenario like this.

15.1.3. Replication Options and Variables

This section describes the options that you can use on slave replication servers. You can specify these options either on the command line or in an option file.

On the master and each slave, you must use the [server-id](#) option to establish a unique replication ID. For each server, you should pick a unique positive integer in the range from 1 to $2^{32} - 1$, and each ID must be different from every other ID. Example: [server-id=3](#)

Options that you can use on the master server for controlling binary logging are described in [Section 5.11.4, “The Binary Log”](#).

Certain options are handled in a special way in order to ensure that the active replication configuration is not inadvertently altered or affected. The options affected are shown in this list:

- [--master-host](#)
- [--master-user](#)
- [--master-password](#)
- [--master-port](#)
- [--master-connect-retry](#)
- [--master-ssl](#)
- [--master-ssl-ca](#)
- [--master-ssl-capath](#)
- [--master-ssl-cert](#)
- [--master-ssl-cipher](#)

- `--master-ssl-key`

In MySQL 5.1.17 and later the use of these options is deprecated. The settings they alter are ignored when `mysqld` is started and a warning will be provided in the `mysqld` log. To configure replication, you must use the `CHANGE MASTER TO ...` statement.

In MySQL 5.1.16 and earlier, these options are ignored if the `master.info` file exists (i.e. when the MySQL server has already previously been configured for replication). If the file exists and these options are present in the `my.cnf` or as options on the command line to `mysqld`, they will be silently ignored and the information in `master.info` used instead.

The `master.info` file format in MySQL 5.1 includes values corresponding to the SSL options. In addition, the file format includes as its first line the number of lines in the file. (See [Section 15.4.2, “Replication Relay and Status Files”](#).) If you upgrade an older server (before MySQL 4.1.1) to a newer version, the new server upgrades the `master.info` file to the new format automatically when it starts. However, if you downgrade a newer server to an older version, you should remove the first line manually before starting the older server for the first time.

If no `master.info` file exists when the slave server starts, it uses the values for those options that are specified in option files or on the command line. This occurs when you start the server as a replication slave for the very first time, or when you have run `RESET SLAVE` and then have shut down and restarted the slave.

If the `master.info` file exists when the slave server starts, the server uses its contents and ignores any options that correspond to the values listed in the file. Thus, if you start the slave server with different values of the startup options that correspond to values in the `master.info` file, the different values have no effect, because the server continues to use the `master.info` file. To use different values, you must either restart after removing the `master.info` file or (preferably) use the `CHANGE MASTER TO` statement to reset the values while the slave is running.

Suppose that you specify this option in your `my.cnf` file:

```
[mysqld]
master-host=some_host
```

The first time you start the server as a replication slave, it reads and uses that option from the `my.cnf` file. The server then records the value in the `master.info` file. The next time you start the server, it reads the master host value from the `master.info` file only and ignores the value in the option file. If you modify the `my.cnf` file to specify a different master host of `some_other_host`, the change still has no effect. You should use `CHANGE MASTER TO` instead.

Because the server gives an existing `master.info` file precedence over the startup options just described, you might prefer not to use startup options for these values at all, and instead specify them by using the `CHANGE MASTER TO` statement. See [Section 12.6.2.1, “CHANGE MASTER TO Syntax”](#).

This example shows a more extensive use of startup options to configure a slave server:

```
[mysqld]
server-id=2
master-host=db-master.mycompany.com
master-port=3306
master-user=pertinax
master-password=freitag
master-connect-retry=60
report-host=db-slave.mycompany.com
```

The following list describes the options and variables used for controlling replication. Many of these options can be reset while the server is running by using the `CHANGE MASTER TO` statement. Others, such as the `--replicate-*` options, can be set only when the slave server starts.

- `--log-slave-updates`

Normally, a slave does not log to its own binary log any updates that are received from a master server. This option tells the slave to log the updates performed by its SQL thread to its own binary log. For this option to have any effect, the slave must also be started with the `--log-bin` option to enable binary logging. `--log-slave-updates` is used when you want to chain replication servers. For example, you might want to set up replication servers using this arrangement:

```
A -> B -> C
```

Here, A serves as the master for the slave B, and B serves as the master for the slave C. For this to work, B must be both a master

and a slave. You must start both A and B with `--log-bin` to enable binary logging, and B with the `--log-slave-updates` option so that updates received from A are logged by B to its binary log.

- `--log-warnings[=level]`

This option causes a server to print more messages to the error log about what it is doing. With respect to replication, the server generates warnings that it succeeded in reconnecting after a network/connection failure, and informs you as to how each slave thread started. This option is enabled by default; to disable it, use `--skip-log-warnings`. Aborted connections are not logged to the error log unless the value is greater than 1.

- `--master-connect-retry=seconds`

The number of seconds that the slave thread sleeps before trying to reconnect to the master in case the master goes down or the connection is lost. The value in the `master.info` file takes precedence if it can be read. If not set, the default is 60. Connection retries are not invoked until the slave times out reading data from the master according to the value of `--slave-net-timeout`. The number of reconnection attempts is limited by the `--master-retry-count` option.

- `--master-host=host_name`

The hostname or IP number of the master replication server. The value in `master.info` takes precedence if it can be read. If no master host is specified, the slave thread does not start.

- `--master-info-file=file_name`

The name to use for the file in which the slave records information about the master. The default name is `master.info` in the data directory.

- `--master-password=password`

The password of the account that the slave thread uses for authentication when it connects to the master. The value in the `master.info` file takes precedence if it can be read. If not set, an empty password is assumed.

- `--master-port=port_number`

The TCP/IP port number that the master is listening on. The value in the `master.info` file takes precedence if it can be read. If not set, the compiled-in setting is assumed (normally 3306).

- `--master-retry-count=count`

The number of times that the slave tries to connect to the master before giving up. Reconnects are attempted at intervals set by `--master-connect-retry` and reconnects are triggered when data reads by the slave time out according to the `--slave-net-timeout` option. The default value is 86400.

- `--master-ssl, --master-ssl-ca=file_name, --master-ssl-capath=directory_name, --master-ssl-cert=file_name, --master-ssl-cipher=cipher_list, --master-ssl-key=file_name`

These options are used for setting up a secure replication connection to the master server using SSL. Their meanings are the same as the corresponding `--ssl, --ssl-ca, --ssl-capath, --ssl-cert, --ssl-cipher, --ssl-key` options that are described in [Section 5.8.7.3, “SSL Command Options”](#). The values in the `master.info` file take precedence if they can be read.

- `--master-user=user_name`

The username of the account that the slave thread uses for authentication when it connects to the master. This account must have the `REPLICATION SLAVE` privilege. The value in the `master.info` file takes precedence if it can be read. If the master username is not set, the name `test` is assumed.

- `--max-relay-log-size=size`

The size at which the server rotates relay log files automatically. For more information, see [Section 15.4.2, “Replication Relay and Status Files”](#). The default size is 1GB.

- `--read-only`

Cause the slave to allow no updates except from slave threads or from users having the `SUPER` privilege. This enables you to ensure that a slave server accepts no updates from clients. This option does not apply to `TEMPORARY` tables.

- `--relay-log=file_name`

The basename for the relay log. The default basename is `host_name-relay-bin`. The server creates relay log files in sequence by adding a numeric suffix to the basename. You can specify the option to create hostname-independent relay log names, or if your relay logs tend to be big (and you don't want to decrease `max_relay_log_size`) and you need to put them in some area different from the data directory, or if you want to increase speed by balancing load between disks.

- `--relay-log-index=file_name`

The name to use for the relay log index file. The default name is `host_name-relay-bin.index` in the data directory, where `host_name` is the name of the slave server.

- `--relay-log-info-file=file_name`

The name to use for the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory.

- `--relay-log-purge={0|1}`

Disable or enable automatic purging of relay logs as soon as they are not needed any more. The default value is 1 (enabled). This is a global variable that can be changed dynamically with `SET GLOBAL relay_log_purge = N`.

- `--relay-log-space-limit=size`

This option places an upper limit on the total size in bytes of all relay logs on the slave. A value of 0 means “no limit.” This is useful for a slave server host that has limited disk space. When the limit is reached, the I/O thread stops reading binary log events from the master server until the SQL thread has caught up and deleted some unused relay logs. Note that this limit is not absolute: There are cases where the SQL thread needs more events before it can delete relay logs. In that case, the I/O thread exceeds the limit until it becomes possible for the SQL thread to delete some relay logs, because not doing so would cause a deadlock. You should not set `--relay-log-space-limit` to less than twice the value of `--max-relay-log-size` (or `--max-binlog-size` if `--max-relay-log-size` is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` is exceeded, but the SQL thread has no relay log to purge and is unable to satisfy the I/O thread. This forces the I/O thread to temporarily ignore `--relay-log-space-limit`.

- `--replicate-do-db=db_name`

Tell the slave to restrict replication to statements where the default database (that is, the one selected by `USE`) is `db_name`. To specify more than one database, use this option multiple times, once for each database. Note that this does not replicate cross-database statements such as `UPDATE some_db.some_table SET foo='bar'` while having selected a different database or no database.

Warning

To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, if you supply a comma separated list then the list will be treated as the name of a single database.

An example of what does not work as you might expect: If the slave is started with `--replicate-do-db=sales` and you issue the following statements on the master, the `UPDATE` statement is *not* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “just check the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

If you need cross-database updates to work, use `--replicate-wild-do-table=db_name.%` instead. See [Section 15.4.3, “How Servers Evaluate Replication Rules”](#).

- `--replicate-do-table=db_name.tbl_name`

Tell the slave thread to restrict replication to the specified table. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-do-db`. See [Section 15.4.3, “How Servers Evaluate Replication Rules”](#).

- `--replicate-ignore-db=db_name`

Tells the slave to not replicate any statement where the default database (that is, the one selected by `USE`) is `db_name`. To specify more than one database to ignore, use this option multiple times, once for each database. You should not use this option if you are using cross-database updates and you do not want these updates to be replicated. See [Section 15.4.3, “How Servers Evaluate Replication Rules”](#).

An example of what does not work as you might expect: If the slave is started with `--replicate-ignore-db=sales` and you issue the following statements on the master, the `UPDATE` statement is replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

Note

In the preceding example the statement is replicated because `--replicate-ignore-db` only applies to the default database (set through the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered.

If you need cross-database updates to work, use `--replicate-wild-ignore-table=db_name.%` instead. See [Section 15.4.3, “How Servers Evaluate Replication Rules”](#).

- `--replicate-ignore-table=db_name.tbl_name`

Tells the slave thread to not replicate any statement that updates the specified table, even if any other tables might be updated by the same statement. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-ignore-db`. See [Section 15.4.3, “How Servers Evaluate Replication Rules”](#).

- `--replicate-rewrite-db=from_name->to_name`

Tells the slave to translate the default database (that is, the one selected by `USE`) to `to_name` if it was `from_name` on the master. Only statements involving tables are affected (not statements such as `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`), and only if `from_name` is the default database on the master. This does not work for cross-database updates. The database name translation is done *before* the `--replicate-*` rules are tested.

If you use this option on the command line and the ‘>’ character is special to your command interpreter, quote the option value. For example:

```
shell> mysqld --replicate-rewrite-db="olddb->newdb"
```

- `--replicate-same-server-id`

To be used on slave servers. Usually you should use the default setting of 0, to prevent infinite loops caused by circular replication. If set to 1, the slave does not skip events having its own server ID. Normally, this is useful only in rare configurations. Cannot be set to 1 if `--log-slave-updates` is used. Note that by default the slave I/O thread does not even write binary log events to the relay log if they have the slave's server id (this optimization helps save disk usage). So if you want to use `--replicate-same-server-id`, be sure to start the slave with this option before you make the slave read its own events that you want the slave SQL thread to execute.

- `--replicate-wild-do-table=db_name.tbl_name`

Tells the slave thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns. Patterns can contain the ‘%’ and ‘_’ wildcard characters, which have the same meaning as for the `LIKE` pattern-matching operator. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates. See [Section 15.4.3, “How Servers Evaluate Replication Rules”](#).

Example: `--replicate-wild-do-table=foo%.bar%` replicates only updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

If the table name pattern is %, it matches any table name and the option also applies to database-level statements (`CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`). For example, if you use `--replicate-wild-do-table=foo%.`, database-level statements are replicated if the database name matches the pattern `foo%`.

To include literal wildcard characters in the database or table name patterns, escape them with a backslash. For example, to replicate all tables of a database that is named `my_own$db`, but not replicate tables from the `my1ownAABCDb` database, you should escape

the ‘_’ and ‘%’ characters like this: `--replicate-wild-do-table=my\_own\%db`. If you're using the option on the command line, you might need to double the backslashes or quote the option value, depending on your command interpreter. For example, with the `bash` shell, you would need to type `--replicate-wild-do-table=my\\_own\\%db`.

- `--replicate-wild-ignore-table=db_name.tbl_name`

Tells the slave thread not to replicate a statement where any table matches the given wildcard pattern. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates. See [Section 15.4.3, “How Servers Evaluate Replication Rules”](#).

Example: `--replicate-wild-ignore-table=foo%.bar%` does not replicate updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

For information about how matching works, see the description of the `--replicate-wild-do-table` option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` as well.

- `--report-host=slave_name`

The hostname or IP number of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server. Leave the value unset if you do not want the slave to register itself with the master. Note that it is not sufficient for the master to simply read the IP number of the slave from the TCP/IP socket after the slave connects. Due to NAT and other routing issues, that IP may not be valid for connecting to the slave from the master or other hosts.

- `--report-port=slave_port_num`

The TCP/IP port number for connecting to the slave, to be reported to the master during slave registration. Set this only if the slave is listening on a non-default port or if you have a special tunnel from the master or other clients to the slave. If you are not sure, do not use this option.

- `--report-password=password`

The account password of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` option is given.

- `--report-user=user_name`

The account username of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` option is given.

- `--show-slave-auth-info`

Display slave usernames and passwords in the output of `SHOW SLAVE HOSTS` on the master server for slaves started with the `--report-user` and `--report-password` options.

- `--skip-slave-start`

Tells the slave server not to start the slave threads when the server starts. To start the threads later, use a `START SLAVE` statement.

- `--slave_compressed_protocol={0|1}`

If this option is set to 1, use compression for the slave/master protocol if both the slave and the master support it. The default is 0 (no compression).

- `--slave-load-tmpdir=file_name`

The name of the directory where the slave creates temporary files. This option is by default equal to the value of the `tmpdir` system variable. When the slave SQL thread replicates a `LOAD DATA INFILE` statement, it extracts the file to be loaded from the relay log into temporary files, and then loads these into the table. If the file loaded on the master is huge, the temporary files on the slave are huge, too. Therefore, it might be advisable to use this option to tell the slave to put temporary files in a directory located in some filesystem that has a lot of available space. In that case, the relay logs are huge as well, so you might also want to use the `--relay-log` option to place the relay logs in that filesystem.

The directory specified by this option should be located in a disk-based filesystem (not a memory-based filesystem) because the temporary files used to replicate `LOAD DATA INFILE` must survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process.

- `--slave-net-timeout=seconds`

The number of seconds to wait for more data from the master before the slave considers the connection broken, aborts the read, and tries to reconnect. The first retry occurs immediately after the timeout. The interval between retries is controlled by the `--master-connect-retry` option and the number of reconnection attempts is limited by the `--master-retry-count` option. The default is 3600 seconds (one hour).

- `--slave-skip-errors=[err_code1,err_code2,...|all]`

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This option tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the option value.

Do not use this option unless you fully understand why you are getting errors. If there are no bugs in your replication setup and client programs, and no bugs in MySQL itself, an error that stops replication should never occur. Indiscriminate use of this option results in slaves becoming hopelessly out of synchrony with the master, with you having no idea why this has occurred.

For error codes, you should use the numbers provided by the error message in your slave error log and in the output of `SHOW SLAVE STATUS`. [Appendix B, Errors, Error Codes, and Common Problems](#), lists server error codes.

You can also (but should not) use the very non-recommended value of `all` to cause the slave to ignore all error messages and keeps going regardless of what happens. Needless to say, if you use `all`, there are no guarantees regarding the integrity of your data. Please do not complain (or file bug reports) in this case if the slave's data is not anywhere close to what it is on the master. *You have been warned.*

Examples:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
```

15.1.4. Common Replication Administration Tasks

Once replication has been started it should execute without requiring much regular administration. Depending on your replication environment, you will want to check the replication status of each slave either periodically, daily, or even more frequently.

MySQL Enterprise

For regular reports regarding the status of your slaves, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

15.1.4.1. Checking Replication Status

The most common task when managing a replication process is to ensure that replication is taking place and that there have been no errors between the slave and the master.

The primary command for this is `SHOW SLAVE STATUS` which you must execute on each slave:

```
mysql> SHOW SLAVE STATUS\G
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: master1
Master_User: root
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000004
Read_Master_Log_Pos: 931
Relay_Log_File: slave1-relay-bin.000056
Relay_Log_Pos: 950
Relay_Master_Log_File: mysql-bin.000004
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 931
```

```

Relay_Log_Space: 1365
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
1 row in set (0.01 sec)

```

The key fields from the status report to examine are:

- **Slave_IO_State** — indicates the current status of the slave. See [Section 6.5.5.5, “Replication Slave I/O Thread States”](#), and [Section 6.5.5.6, “Replication Slave SQL Thread States”](#), for more information.
- **Slave_IO_Running** — shows whether the IO thread for the reading the master's binary log is running.
- **Slave_SQL_Running** — shows whether the SQL thread for the executing events in the relay log is running.
- **Last_Error** — shows the last error registered when processing the relay log. Ideally this should be blank, indicating no errors.
- **Seconds_Behind_Master** — shows the number of seconds that the slave SQL thread is behind processing the master binary log. A high number (or an increasing one) can indicate that the slave is unable to cope with the large number of queries from the master.

On the master, you can check the status of slaves by examining the list of running processes. Slaves execute the **Binlog Dump** command:

```

mysql> SHOW PROCESSLIST \G;
***** 4. row *****
   Id: 10
  User: root
 Host: slave1:58371
   db: NULL
Command: Binlog Dump
   Time: 777
  State: Has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL

```

Because it is the slave that drives the core of the replication process, very little information is available in this report.

If you have used the **--report-host** option, then the **SHOW SLAVE HOSTS** statement will show basic information about connected slaves:

```

mysql> SHOW SLAVE HOSTS;
+-----+-----+-----+-----+-----+
| Server_id | Host | Port | Rpl_recovery_rank | Master_id |
+-----+-----+-----+-----+-----+
| 10 | slave1 | 3306 | 0 | 1 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

The output includes the ID of the slave server, the value of the **--report-host** option, the connecting port, master ID and the priority of the slave for receiving binary log updates.

15.1.4.2. Pausing Replication on the Slave

You can stop and start the replication of statements on the slave using the **STOP SLAVE** and **START SLAVE** commands.

To stop execution of the binary log from the master, use **STOP SLAVE**:

```
mysql> STOP SLAVE;
```

When execution is stopped, the slave does not read the binary log from the master (the **IO_THREAD**) and stops processing events from the relay log that have not yet been executed (the **SQL_THREAD**). You can pause either the IO or SQL threads individually by specifying the thread type. For example:


```
mysql> STOP SLAVE IO_THREAD;
```

Stopping the SQL thread can be useful if you want to perform a backup or other task on a slave that only processes events from the master. The IO thread will continue to be read from the master, but not executed, which will make it easier for the slave to catch up when you start slave operations again.

Stopping the IO thread will allow the statements in the relay log to be executed up until the point where the relay log has ceased to receive new events. Using this option can be useful when you want to pause execution to allow the slave to catch up with events from the master, when you want to perform administration on the slave but also ensure you have the latest updates to a specific point. This method can also be used to pause execution on the slave while you conduct administration on the master while ensuring that there is not a massive backlog of events to be executed when replication is started again.

To start execution again, use the `START SLAVE` statement:

```
mysql> START SLAVE;
```

If necessary, you can start either the `IO_THREAD` or `SQL_THREAD` threads individually.

15.2. Replication Solutions

Replication can be used in many different environments for a range of purposes. In this section you will find general notes and advice on using replication for specific solution types.

For information on using replication in a backup environment, including notes on the setup, backup procedure, and files to backup, see [Section 15.2.1, “Using Replication for Backups”](#).

For advice and tips on using different storage engines on the master and slaves, see [Section 15.2.2, “Using Replication with different Master and Slave Storage Engines”](#).

Using replication as a scale-out solution requires some changes in the logic and operation of applications that use the solution. See [Section 15.2.3, “Using Replication for Scale-out”](#).

For performance or data distribution reasons you may want to replicate different databases to different replication slaves. See [Section 15.2.4, “Replicating Different Databases to Different Slaves”](#).

As the number of replication slaves increases, the load on the master can increase (because of the need to replicate the binary log to each slave) and lead to a reduction in performance of the master. For tips on improving your replication performance, including using a single secondary server as an replication master, see [Section 15.2.5, “Improving Replication Performance”](#).

For guidance on switching masters, or converting slaves into masters as part of an emergency failover solution, see [Section 15.2.6, “Switching Masters During Failover”](#).

To secure your replication communication you can encrypt the communication channel by using SSL to exchange data. Step-by-step instructions can be found in [Section 15.2.7, “Setting up Replication using SSL”](#).

15.2.1. Using Replication for Backups

You can use replication as a backup solution by replicating data from the master to a slave, and then backing up the data slave. Because the slave can be paused and shutdown without affecting the running operation of the master you can produce an effective snapshot of 'live' data that would otherwise require a shutdown of the master database.

How you back up the database will depend on the size of the database and whether you are backing up only the data, or the data and the replication slave state so that you can rebuild the slave in the event of failure. There are therefore two choices:

If you are using replication as a solution to enable you to backup the data on the master, and the size of your database is not too large, then the `mysqldump` tool may be suitable. See [Section 15.2.1.1, “Backing up using mysqldump”](#).

For larger databases, where `mysqldump` would be impractical or inefficient, you can backup the raw data files instead. Using the raw data files option also means that you can backup the binary and relay logs that will enable you to recreate the slave in the event of a slave failure. For more information, see [Section 15.2.1.2, “Backing up raw data”](#).

15.2.1.1. Backing up using `mysqldump`

Using `mysqldump` to create a copy of the database enables you to capture all of the data in the database in a format that allows the information to be imported into another instance of MySQL. Because the format of the information is SQL statements the file can easily be distributed and applied to running servers in the event that you need access to the data in an emergency. However, if the size of your data set is very large then `mysqldump` may be impractical.

When using `mysqldump` you should stop the slave before starting the dump process to ensure that the dump contains a consistent set of data:

1. Stop the slave from processing requests. You can either stop the slave completely using `mysqladmin`:

```
shell> mysqladmin stop-slave
```

Alternatively, you can stop processing the relay log files by stopping the replication SQL thread. Using this method will allow the binary log data to be transferred. Within busy replication environments this may speed up the catch-up process when you start the slave processing again:

```
shell> mysql -e 'STOP SLAVE SQL_THREAD;'
```

2. Run `mysqldump` to dump your databases. You may either select databases to be dumped, or dump all databases. For more information see [Section 7.13, “mysqldump — A Database Backup Program”](#). For example, to dump all databases:

```
shell> mysqldump --all-databases >fulldb.dump
```

3. Once the dump has completed, start slave operations again:

```
shell> mysqladmin start-slave
```

In the preceding example you may want to add login credentials (username, password) to the commands, and bundle the process up into a script that you can run automatically each day.

If you use this approach, make sure you monitor the slave replication process to ensure that the time taken to run the backup in this way is not affecting the slaves ability to keep up with events from the master. See [Section 15.1.4.1, “Checking Replication Status”](#). If the slave is unable to keep up you may want to add another server and distribute the backup process. For an example of how to configure this scenario, see [Section 15.2.4, “Replicating Different Databases to Different Slaves”](#).

15.2.1.2. Backing up raw data

To guarantee the integrity of the files that are copied, backing up the raw data files on your MySQL replication slave should take place while your slave server is shutdown. If the MySQL server is still running then background tasks, particularly with storage engines with background processes such as InnoDB, may still be updating the database files. With InnoDB, these problems should be resolved during crash recovery, but since the slave server can be shutdown during the backup process without affecting the execution of the master it makes sense to take advantage of this facility.

To shutdown the server and backup the files:

1. Shutdown the slave MySQL server:

```
shell> mysqladmin shutdown
```

2. Copy the data files. You can use any suitable copying or archive utility, including `cp`, `tar` or `WinZip`:

```
tar cf /tmp/dbbackup.tar ./data
```

3. Start up the `mysqld` process again:

```
shell> mysqld_safe &
```

Under Windows:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld"
```

Normally you should back up the entire data folder for the slave MySQL server. If you want to be able to restore the data and operate as a slave (for example, in the event of failure of the slave), then when you back up the slave's data, you should back up the slave status files, `master.info` and `relay.info`, along with the relay log files. These files are needed to resume replication after you restore the slave's data.

If you lose the relay logs but still have the `relay-log.info` file, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. Of course, this requires that the binary logs still exist on the master server.

If your slave is subject to replicating `LOAD DATA INFILE` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication of any interrupted `LOAD DATA INFILE` operations. The directory location is specified using the `--slave-load-tmpdir` option. If this option is not specified, the directory location is the value of the `tmpdir` system variable.

15.2.2. Using Replication with different Master and Slave Storage Engines

The replication process does not care if the source table on the master and the replicated table on the slave use different engine types. In fact, the system variables `storage_engine` and `table_type` are not replicated.

This provides a number of advantages in the replication process in that you can take advantage of different engine types for different replication scenarios. For example, in a typical scaleout scenario (see [Section 15.2.3, “Using Replication for Scale-out”](#)), you want to use `InnoDB` tables on the master to take advantage of the transactional functionality, but use `MyISAM` on the slaves where transaction support is not required because the data is only read. When using replication in a data logging environment you may want to use the `Archive` storage engine on the slave.

Setting up different engines on the master and slave depends on how you set up the initial replication process:

- If you used `mysqldump` to create the database snapshot on your master then you could edit the dump text to change the engine type used on each table.

Another alternative for `mysqldump` is to disable engine types that you do not want to use on the slave before using the dump to build the data on the slave. For example, you can add the `--skip-innodb` option on your slave to disable the `InnoDB` engine. If a specific engine does not exist, MySQL will use the default engine type, usually `MyISAM`. If you want to disable further engines in this way, you may want to consider building a special binary to be used on the slave that only supports the engines you want.

- If you are using raw data files for the population of the slave, you will be unable to change the initial table format. Instead, use `ALTER TABLE` to change the table types after the slave has been started.
- For new master/slave replication setups where there are currently no tables on the master, avoid specifying the engine type when creating new tables.

If you are already running a replication solution and want to convert your existing tables to another engine type, follow these steps:

1. Stop the slave from running replication updates:

```
mysql> STOP SLAVE;
```

This will enable you to change engine types without interruptions.

2. Execute an `ALTER TABLE ... Engine='enginetype'` for each table where you want to change the engine type.
3. Start the slave replication process again:

```
mysql> START SLAVE;
```

Although the `storage_engine` and `table_type` variables are not replicated, be aware that `CREATE TABLE` and `ALTER TABLE` statements that include the engine specification will be correctly replicated to the slave. For example, if you have a CSV table and you execute:

```
mysql> ALTER TABLE csvtable Engine='MyISAM';
```

The above statement will be replicated to the slave and the engine type on the slave will be converted to [MyISAM](#), even if you have previously changed the table type on the slave to an engine other than CSV. If you want to retain engine differences on the master and slave, you should be careful to use the [storage_engine](#) variable on the master when creating a new table. For example, instead of:

```
mysql> CREATE TABLE tablea (columna int) Engine=MyISAM;
```

Use this format:

```
mysql> SET storage_engine=MyISAM;
mysql> CREATE TABLE tablea (columna int);
```

When replicated, the [storage_engine](#) variable will be ignored, and the [CREATE TABLE](#) statement will be executed with the slave's default engine type.

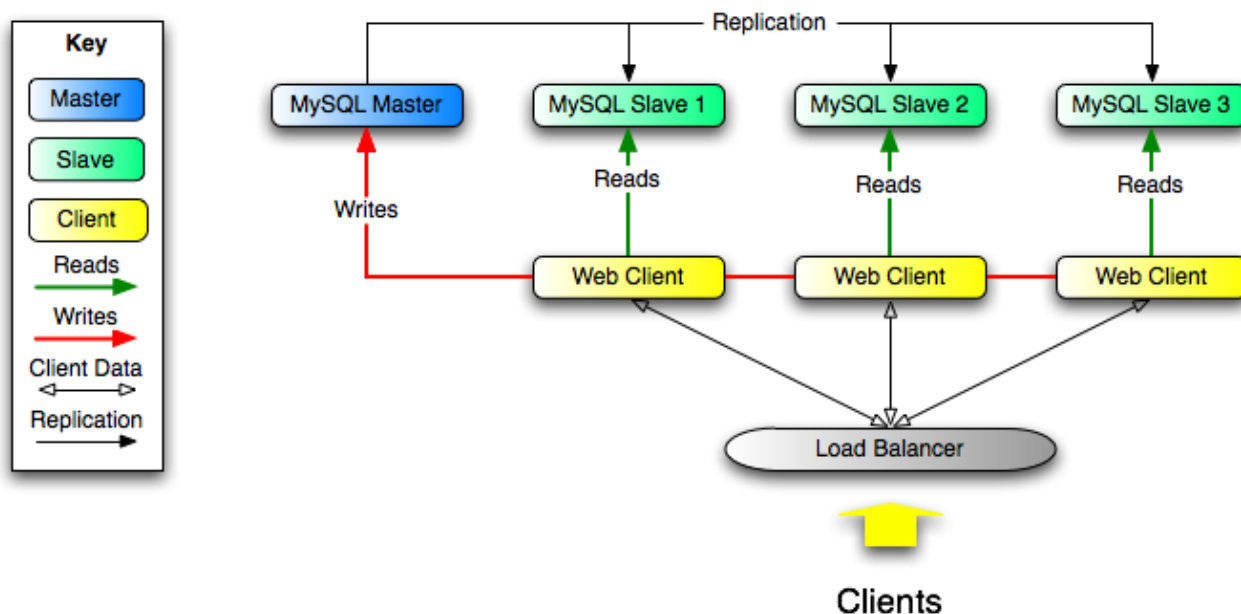
15.2.3. Using Replication for Scale-out

You can use replication as a scale-out solution, i.e. where you want to split up the load of database queries across multiple database servers, within some reasonable limitations.

Because replication works from the distribution of one master to one or more slaves, using replication for scaleout works best in an environment where you have a high number of reads and low number of writes/updates. Most websites fit into this category, where users are browsing the website, reading articles, posts, or viewing products. Updates only occur during session management, or when making a purchase or adding a comment/message to a forum.

Replication in this situation enables you to distribute the reads over the replication slaves, while still allowing your web servers to communicate with the replication master when a write is required. You can see a sample replication layout for this scenario in [Figure 15.1](#), “Using replication to improve the performance during scaleout”.

Figure 15.1. Using replication to improve the performance during scaleout



If the part of your code that is responsible for database access has been properly abstracted/modularized, converting it to run with a replicated setup should be very smooth and easy. Change the implementation of your database access to send all writes to the master, and to send reads to either the master or a slave. If your code does not have this level of abstraction, setting up a replicated system gives you the opportunity and motivation to clean it up. Start by creating a wrapper library or module that implements the following functions:

- `safe_writer_connect()`

- `safe_reader_connect()`
- `safe_reader_statement()`
- `safe_writer_statement()`

`safe_` in each function name means that the function takes care of handling all error conditions. You can use different names for the functions. The important thing is to have a unified interface for connecting for reads, connecting for writes, doing a read, and doing a write.

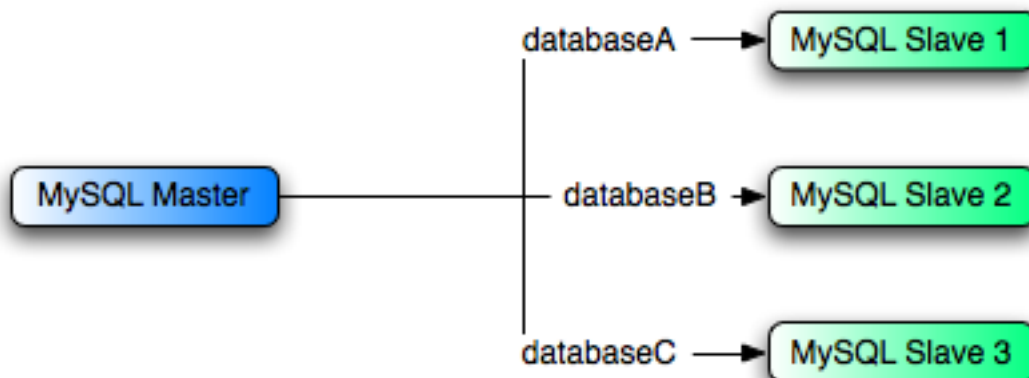
Then convert your client code to use the wrapper library. This may be a painful and scary process at first, but it pays off in the long run. All applications that use the approach just described are able to take advantage of a master/slave configuration, even one involving multiple slaves. The code is much easier to maintain, and adding troubleshooting options is trivial. You need modify only one or two functions; for example, to log how long each statement took, or which statement among those issued gave you an error.

If you have written a lot of code, you may want to automate the conversion task by using the `replace` utility that comes with standard MySQL distributions, or write your own conversion script. Ideally, your code uses consistent programming style conventions. If not, then you are probably better off rewriting it anyway, or at least going through and manually regularizing it to use a consistent style.

15.2.4. Replicating Different Databases to Different Slaves

There may be situations where you have a single master and want to replicate different databases to different slaves. For example, you may want to distribute different sales data to different departments to help spread the load during data analysis. A sample of this layout is shown in [Figure 15.2, “Using replication to replicate separate DBs to multiple hosts”](#).

Figure 15.2. Using replication to replicate separate DBs to multiple hosts



You can achieve this separation by configuring the master and slaves as normal, and then limiting the binary log statements that each slave processes by using the `replicate-wild-do-table` configuration option on each slave.

For example, to support the separation as shown in [Figure 15.2, “Using replication to replicate separate DBs to multiple hosts”](#), you would configure each slave as follows before enabling replication using `START SLAVE`:

- MySQL Slave 1 should have the following configuration options:

```
replicate-wild-do-table=sales.%
replicate-wild-do-table=finance.%
```

- MySQL Slave 2 should have the following configuration option:

```
replicate-wild-do-table=support.%
```

- MySQL Slave 3 should have the following configuration option:

```
replicate-wild-do-table=service.%
```

If you have data that needs to be synchronized to the slaves before replication starts, you have a number of options:

- Synchronize all the data to each slave, and delete the databases and/or tables that you do not want to keep.
- Use `mysqldump` to create a separate dump file for each database and load the appropriate dump file on each slave.
- Use a raw data file dump and include only the specific files and databases that you need for each slave. This option will not work with InnoDB databases unless you use the `innodb_file_per_table` option.

Each slave in this configuration will transfer to the entire binary log from the master, but will only execute the events within the binary log that apply to the configured databases and tables.

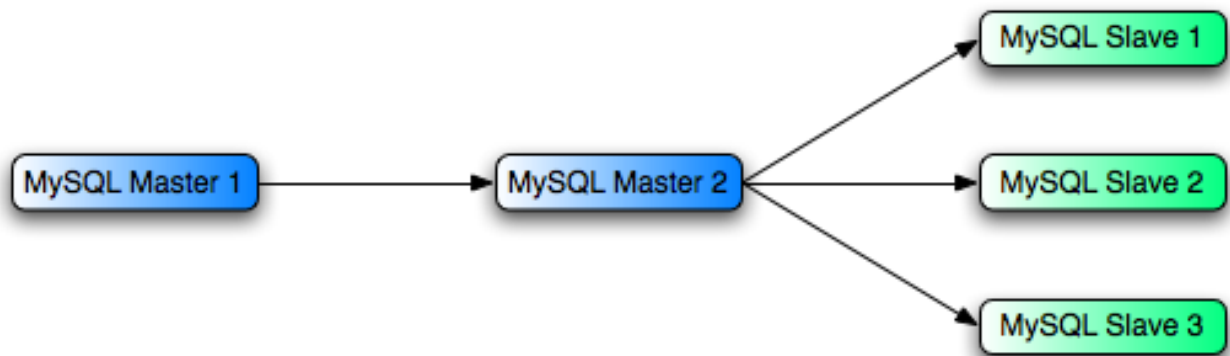
15.2.5. Improving Replication Performance

As the number of slaves connecting to a master increases, the load, although minimal, also increases, as each slave uses up a client connection to the master. Also, as each slave must receive a full copy of the master binary log, the network load on the master may also increase and start to create a bottleneck.

If you are using a large number of slaves connected to one master, and that master is also busy processing requests (for example, as part of a scaleout solution), then you may want to improve the performance of the replication process.

One way to improve the performance of the replication process is to create a deeper replication structure that enables the master to replicate to only one slave, and for the remaining slaves to connect to this primary slave for their individual replication requirements. A sample of this structure is shown in [Figure 15.3, “Using an additional replication host to improve performance”](#).

Figure 15.3. Using an additional replication host to improve performance



For this to work, you must configure the MySQL instances as follows:

- Master 1 is the primary master where all changes and updates are written to the database. Binary logging should be enabled on this machine.
- Master 2 is the slave to the Master 1 that provides the replication functionality to the remainder of the slaves in the replication structure. Master 2 is the only machine allowed to connect to Master 1. Master 2 also has binary logging enabled, and the `-log-slave-updates` option so that replication instructions from Master 1 are also written to Master 2's binary log so that they can then be replicated to the true slaves.
- Slave 1, Slave 2, and Slave 3 act as slaves to Master 2, and replicate the information from Master 2, which is really the data logged

on Master 1.

The above solution reduces the client load and the network interface load on the primary master, which should improve the overall performance of the primary master when used as a direct database solution.

If your slaves are having trouble keeping up with the replication process on the master then there are a number of options available:

- If possible, you should put the relay logs and the data files on different physical drives. To do this, use the `--relay-log` option to specify the location of the relay log.
- If the slaves are significantly slower than the master, then you may want to divide up the responsibility for replicating different databases to different slaves. See [Section 15.2.4, “Replicating Different Databases to Different Slaves”](#).
- If your master makes use of transactions and you are not concerned about transaction support on your slaves, then use [MyISAM](#) or another non-transactional engine. See [Section 15.2.2, “Using Replication with different Master and Slave Storage Engines”](#).
- If your slaves are not acting as masters, and you have a potential solution in place to ensure that you can bring up a master in the event of failure, then you can switch off `--log-slave-updates`. This prevents 'dumb' slaves from also logging events they have executed into their own binary log.

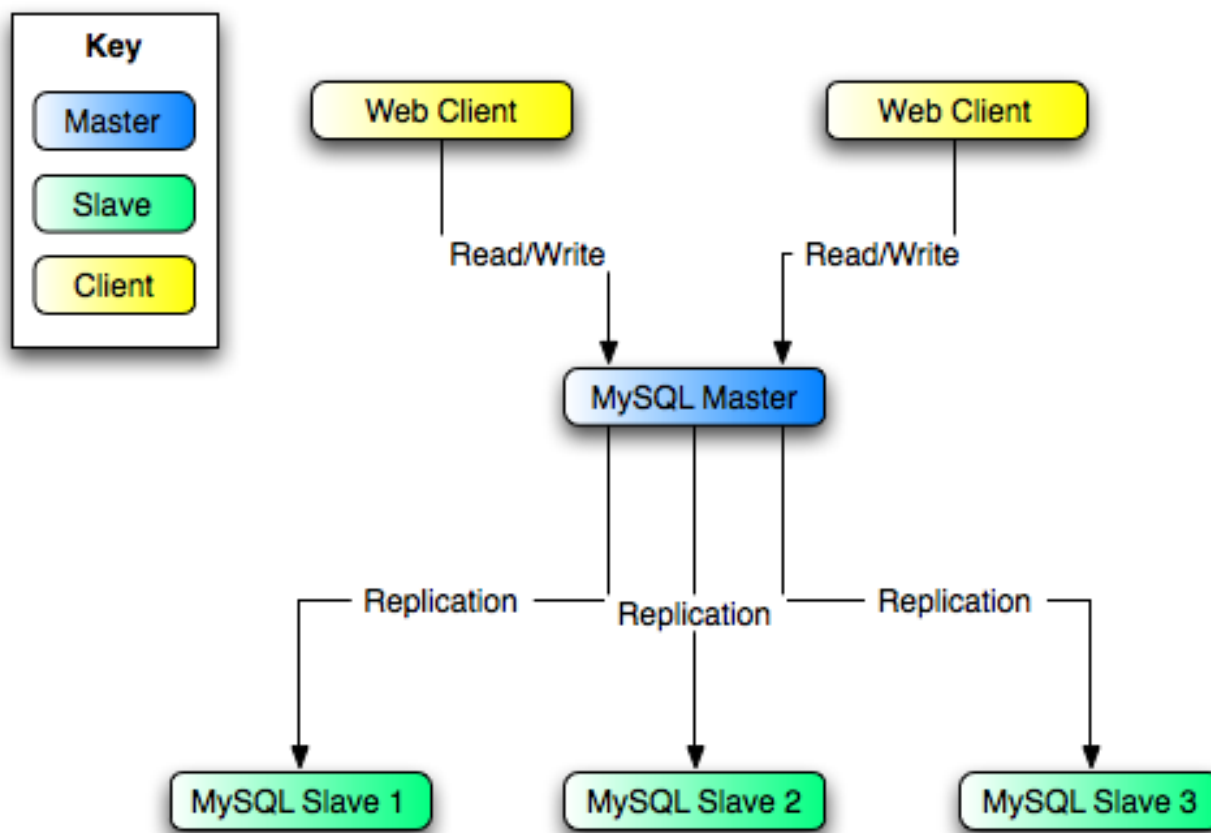
15.2.6. Switching Masters During Failover

There is currently no official solution for providing failover between master and slaves in the event of a failure. With the currently available features, you would have to set up a master and a slave (or several slaves), and to write a script that monitors the master to check whether it is up. Then instruct your applications and the slaves to change master in case of failure.

Remember that you can tell a slave to change its master at any time, using the `CHANGE MASTER TO` statement. The slave will not check whether the databases on the master are compatible with the slave, it will just start executing events from the specified log and position on the new master. In a failover situation all the servers in the group are probably executing the same events from the same binary log, so changing the source of the events should not affect the database structure or integrity providing you are careful.

Run your slaves with the `--log-bin` option and without `--log-slave-updates`. In this way, the slave is ready to become a master as soon as you issue `STOP SLAVE; RESET MASTER`, and `CHANGE MASTER TO` statement on the other slaves. For example, assume that you have the structure shown in [Figure 15.4, “Redundancy using replication, initial structure”](#).

Figure 15.4. Redundancy using replication, initial structure



In this diagram, the **MySQL Master** holds the master database, the **MySQL Slave** computers are replication slaves, and the **Web Client** machines are issuing database reads and writes. Web clients that issue only reads (and would normally be connected to the slaves) are not shown, as they do not need to switch to a new server in the event of failure. For a more detailed example of a read/write scaleout replication structure, see [Section 15.2.3, “Using Replication for Scale-out”](#).

Each **MySQL Slave** (**Slave 1**, **Slave 2**, and **Slave 3**) are slaves running with `--log-bin` and without `--log-slave-updates`. Because updates received by a slave from the master are not logged in the binary log unless `--log-slave-updates` is specified, the binary log on each slave is empty initially. If for some reason **MySQL Master** becomes unavailable, you can pick one of the slaves to become the new master. For example, if you pick **Slave 1**, all **Web Clients** should be redirected to **Slave 1**, which will log updates to its binary log. **Slave 2** and **Slave 3** should then replicate from **Slave 1**.

The reason for running the slave without `--log-slave-updates` is to prevent slaves from receiving updates twice in case you cause one of the slaves to become the new master. Suppose that **Slave 1** has `--log-slave-updates` enabled. Then it will write updates that it receives from **Master** to its own binary log. When **Slave 2** changes from **Master** to **Slave 1** as its master, it may receive updates from **Slave 1** that it has already received from **Master**.

Make sure that all slaves have processed any statements in their relay log. On each slave, issue `STOP SLAVE IO_THREAD`, then check the output of `SHOW PROCESSLIST` until you see `Has read all relay log`. When this is true for all slaves, they can be reconfigured to the new setup. On the slave **Slave 1** being promoted to become the master, issue `STOP SLAVE` and `RESET MASTER`.

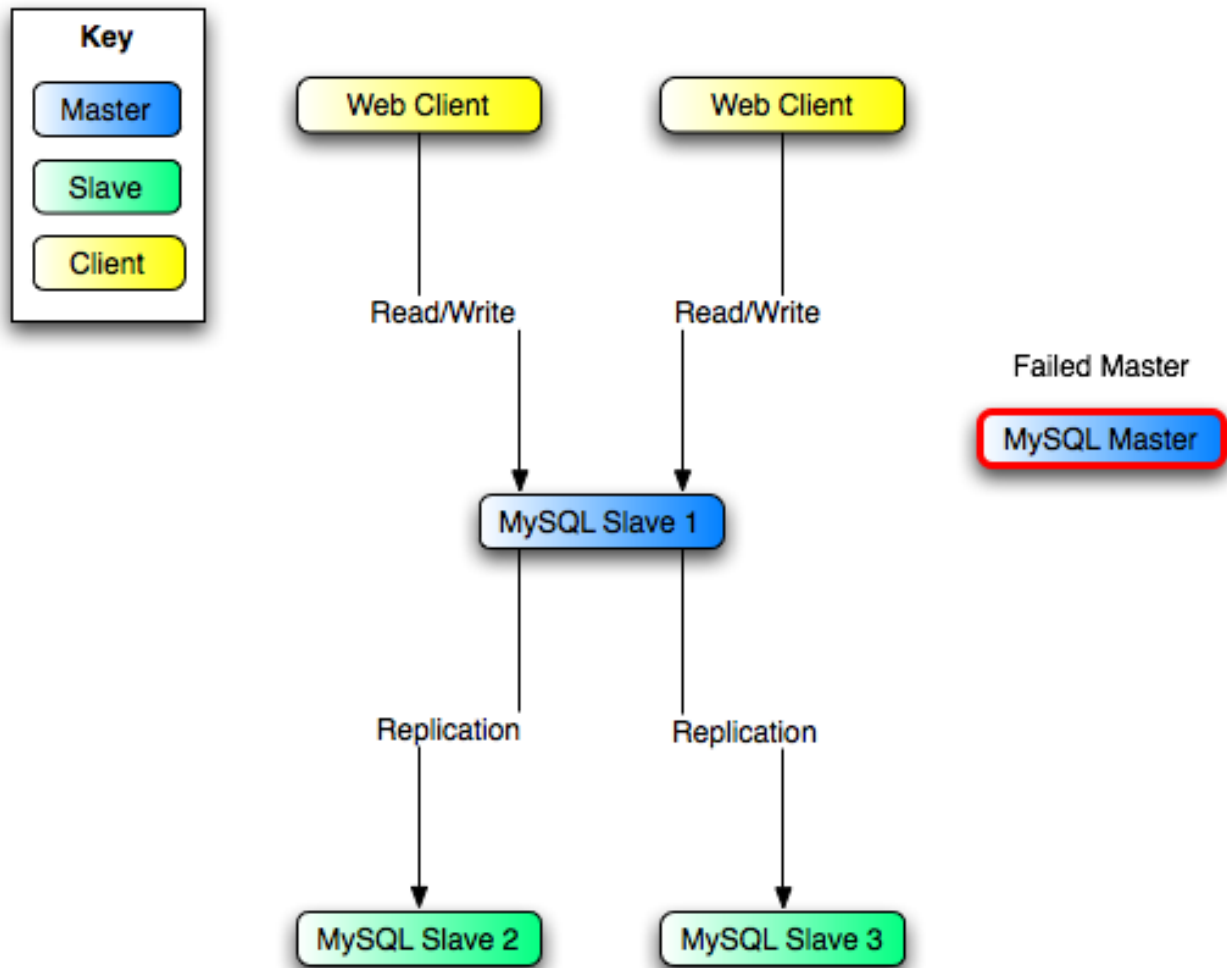
On the other slaves **Slave 2** and **Slave 3**, use `STOP SLAVE` and `CHANGE MASTER TO MASTER_HOST='Slave1'` (where `'Slave1'` represents the real hostname of **Slave 1**). To `CHANGE MASTER`, add all information about how to connect to **Slave 1** from **Slave 2** or **Slave 3** (`user`, `password`, `port`). In `CHANGE MASTER`, there is no need to specify the name of **Slave 1**'s binary log or binary log position to read from: We know it is the first binary log and position 4, which are the defaults for `CHANGE MASTER`. Finally, use `START SLAVE` on **Slave 2** and **Slave 3**.

Once the new replication is in place, you will then need to instruct each **Web Client** to direct their statements to **Slave 1**. From that point on, all updates statements sent by **Web Client** to **Slave 1** are written to the binary log of **Slave 1**, which then contains

every update statement sent to **Slave 1** since **Master** died.

The resulting server structure is shown in [Figure 15.5, “Redundancy using replication, after master failure”](#).

Figure 15.5. Redundancy using replication, after master failure



When **Master** is up again, you must issue on it the same **CHANGE MASTER** as that issued on **Slave 2** and **Slave 3**, so that **Master** becomes a slave of **S1** and picks up each **Web Client** writes that it missed while it was down.

To make **Master** a master again (because it is the most powerful machine, for example), use the preceding procedure as if **Slave 1** was unavailable and **Master** was to be the new master. During this procedure, do not forget to run **RESET MASTER** on **Master** before making **Slave 1**, **Slave 2**, and **Slave 3** slaves of **Master**. Otherwise, they may pick up old **Web Client** writes from before the point at which **Master** became unavailable.

Note that there is no synchronization between the different slaves to a master. Some slaves might be ahead of others. This means that the concept outlined in the previous example might not work. In practice, however, the relay logs of different slaves will most likely not be far behind the master, so it would work, anyway (but there is no guarantee).

A good way to keep your applications informed as to the location of the master is by having a dynamic DNS entry for the master. With **bind** you can use **nsupdate** to dynamically update your DNS.

15.2.7. Setting up Replication using SSL

Setting up replication using an SSL connection is similar to setting up a server and client using SSL. You will need to obtain (or create) a suitable security certificate that you can use on the master, and a similar certificate (from the same certificate authority) on each slave.

To use SSL for encrypting the transfer of the binary log required during replication you must first set up the master to support SSL network connections. If the master does not support SSL connections (because it has not been compiled or configured for SSL), then replication through an SSL connection will not be possible.

For more information on setting up a server and client for SSL connectivity, see [Section 5.8.7.2, “Using SSL Connections”](#).

To enable SSL on the master you will need to create or obtain suitable certificates and then add the following configuration options to the master's configuration within the `mysqld` section:

```
ssl-ca=cacert.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

The options are as follows:

- `ssl-ca` identifies the Certificate Authority (CA) certificate.
- `ssl-cert` identifies the server public key. This can be sent to the client and authenticated against the CA certificate that it has.
- `ssl-key` identifies the server private key.

On the slave, you have two options available for setting the SSL information. You can either add the slaves certificates to the `client` section of the slave configuration file, or you can explicitly specify the SSL information using the `CHANGE MASTER` statement.

Using the former option, add the following lines to the `client` section of the slave configuration file:

```
[client]
ssl-ca=cacert.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

Restart the slave server, using the `--skip-slave` to prevent the slave from connecting to the master. Use `CHANGE MASTER` to specify the master configuration, using the `master_ssl` option to enable SSL connectivity:

```
mysql> CHANGE MASTER TO \
  MASTER_HOST='master_hostname', \
  MASTER_USER='replicate', \
  MASTER_PASSWORD='password', \
  MASTER_SSL=1;
```

To specify the SSL certificate options during the `CHANGE MASTER` command, append the SSL options:

```
CHANGE MASTER TO \
  MASTER_HOST='master_hostname', \
  MASTER_USER='replicate', \
  MASTER_PASSWORD='password', \
  MASTER_SSL=1, \
  MASTER_SSL_CA = 'ca_file_name', \
  MASTER_SSL_CAPATH = 'ca_directory_name', \
  MASTER_SSL_CERT = 'cert_file_name', \
  MASTER_SSL_KEY = 'key_file_name';
```

Once the master information has been updated, start the slave replication process:

```
mysql> START SLAVE;
```

You can use the `SHOW SLAVE STATUS` to confirm that SSL connection has been completed.

For more information on the `CHANGE MASTER TO` syntax, see [Section 12.6.2.1, “CHANGE MASTER TO Syntax”](#).

If you want to enforce SSL connections to be used during replication, then create a user with the `REPLICATION SLAVE` privilege and use the `REQUIRE_SSL` option for that user. For example:

```
mysql> GRANT REPLICATION SLAVE ON *.*
```

```
-> TO 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass' REQUIRE SSL;
```

15.3. Replication Notes and Tips

15.3.1. Replication Features and Issues

In general, replication compatibility at the SQL level requires that any features used be supported by both the master and the slave servers. If you use a feature on a master server that is available only as of a given version of MySQL, you cannot replicate to a slave that is older than that version. Such incompatibilities are likely to occur between series, so that, for example, you cannot replicate from MySQL 5.1 to 5.0. However, these incompatibilities also can occur for within-series replication. For example, the `SLEEP()` function is available in MySQL 5.0.12 and up. If you use this function on the master server, you cannot replicate to a slave server that is older than MySQL 5.0.12.

If you are planning to use replication between 5.1 and a previous version of MySQL you should consult the edition of the MySQL Reference Manual corresponding to the earlier release series for information regarding the replication characteristics of that series.

The following sections provide details about what is supported and what is not, and about specific issues and situations that may occur when replicating certain statements. Additional **InnoDB**-specific information about replication is given in [Section 13.5.6.5, “InnoDB and MySQL Replication”](#).

With MySQL's classic statement-based replication, there may be issues with replicating stored routines or triggers. You can avoid these issues by using MySQL's row-based replication instead. For a detailed list of issues, see [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#). For a description of row-based replication, see [Section 15.1.2, “Replication Formats”](#).

15.3.1.1. Replication and **AUTO_INCREMENT**

Replication of **AUTO_INCREMENT**, `LAST_INSERT_ID()`, and **TIMESTAMP** values is done correctly, subject to the following exceptions.

A stored procedure that uses `LAST_INSERT_ID()` does not replicate properly using statement-based binary logging. This limitation is lifted in MySQL 5.1.12.

Adding an **AUTO_INCREMENT** column to a table with `ALTER TABLE` might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an **AUTO_INCREMENT** number. Assuming that you want to add an **AUTO_INCREMENT** column to the table `t1`, the following statements produce a new table `t2` identical to `t1` but with an **AUTO_INCREMENT** column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

This assumes that the table `t1` has columns `col1` and `col2`.

Important

To guarantee the same ordering on both master and slave, *all* columns of `t1` must be referenced in the `ORDER BY` clause.

The instructions just given are subject to the limitations of `CREATE TABLE ... LIKE`: Foreign key definitions are ignored, as are the **DATA DIRECTORY** and **INDEX DIRECTORY** table options. If a table definition includes any of those characteristics, create `t2` using a `CREATE TABLE` statement that is identical to the one used to create `t1`, but with the addition of the **AUTO_INCREMENT** column.

Regardless of the method used to create and populate the copy having the **AUTO_INCREMENT** column, the final step is to drop the original table and then rename the copy:

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

See also [Section B.1.7.1, “Problems with ALTER TABLE”](#).

15.3.1.2. Replication and Character Sets

The following applies to replication between MySQL servers that use different character sets:

1. If the master uses MySQL 4.1, you must *always* use the same *global* character set and collation on the master and the slave, regardless of the MySQL version running on the slave. (These are controlled by the `--character-set-server` and `--collation-server` options.) Otherwise, you may get duplicate-key errors on the slave, because a key that is unique in the master character set might not be unique in the slave character set. Note that this is not a cause for concern when master and slave are both MySQL 5.0 or later.
2. If the master is older than MySQL 4.1.3, the character set of any client should never be made different from its global value because this character set change is not known to the slave. In other words, clients should not use `SET NAMES`, `SET CHARACTER SET`, and so forth. If both the master and the slave are 4.1.3 or newer, clients can freely set session values for character set variables because these settings are written to the binary log and so are known to the slave. That is, clients can use `SET NAMES` or `SET CHARACTER SET` or can set variables such as `collation_client` or `collation_server`. However, clients are prevented from changing the *global* value of these variables; as stated previously, the master and slave must always have identical global character set values.
3. If you have databases on the master with character sets that differ from the global `character_set_server` value, you should design your `CREATE TABLE` statements so that tables in those databases do not implicitly rely on the database default character set. A good workaround is to state the character set and collation explicitly in `CREATE TABLE` statements.

15.3.1.3. Replication of `CREATE TABLE ... SELECT` Statements

The following rules and decisions are applied when a `CREATE TABLE ... SELECT` statement is replicated:

- All `CREATE TABLE ... SELECT` statements do implicit commit.
- If there are no failures, then all `CREATE TABLE ... SELECT` statements are replicated as follows:
 - For `STATEMENT` and `MIXED` format, as the `CREATE TABLE ... SELECT` statement itself.
 - For `ROW` format, as a `CREATE TABLE` statement followed by binwrite events.
- Requirements for `CREATE TABLE t2 SELECT ...` using both transactional and non-transactional tables:
 - For `STATEMENT`, `MIXED`, and `ROW` formats:
 - If the table already exists, then the statement does nothing on the master, and only the implicit commit is logged.
 - If other execution failure (and thus `t2` did not exist), then the table is never created on master and only the implicit commit is logged.
- Requirements for `CREATE TABLE IF NOT EXISTS t2 SELECT ...`
 - For `STATEMENT` and `MIXED` format:

If execution failure, then statement is logged with error code.
 - For `ROW` format when `t2` is transactional:
 - If table already exists, then
 - The `CREATE TABLE` part of the statement is not logged.
 - All applied rows are logged.
 - The implicit commit is logged.
 - If other failure in creating table, then only the implicit commit is logged.
 - If failure in selecting or inserting (but create succeeded), then the table is dropped on master and only the implicit commit is logged.
 - For `ROW` format when `t2` is non-transactional:

- If table already exists, then:
 - the `CREATE TABLE` part of the statement is not logged.
 - All applied rows are logged.
 - The implicit commit is logged.
- If other failure in creating table, then only the implicit commit is logged.
- If failure in selecting or inserting (but create succeeded), then:
 - The `CREATE TABLE` part of the statement is logged.
 - All applied rows are logged.
 - The implicit commit is logged.

15.3.1.4. Replication and `DIRECTORY` Statements

If a `DATA DIRECTORY` or `INDEX DIRECTORY` table option is used in a `CREATE TABLE` statement on the master server, the table option is also used on the slave. This can cause problems if no corresponding directory exists in the slave host filesystem or if it exists but is not accessible to the slave server. MySQL supports an `sql_mode` option called `NO_DIR_IN_CREATE`. If the slave server is run with this SQL mode enabled, it ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options when replicating `CREATE TABLE` statements. The result is that `MyISAM` data and index files are created in the table's database directory.

15.3.1.5. Replication of Invoked Features

Replication of invoked features such as scheduled events, user-defined functions (UDFs), stored routines (including both stored procedures and stored functions), and triggers was re-implemented in MySQL 5.1.18 to provide the following characteristics:

- The effects of the feature are always replicated.
- The following statements are replicated using statement-based replication:
 - `CREATE EVENT`
 - `ALTER EVENT`
 - `DROP EVENT`
 - `CREATE PROCEDURE`
 - `DROP PROCEDURE`
 - `CREATE FUNCTION`
 - `DROP FUNCTION`
 - `CREATE TRIGGER`
 - `DROP TRIGGER`

However, the *effects* of features created, modified, or dropped using these statements are replicated using row-based replication.

- In the case of `CREATE EVENT` and `ALTER EVENT`:
 - The status of the event is set to `SLAVESIDE_DISABLED` on the slave regardless of the state specified (this does not apply to `DROP EVENT`).
 - The master on which the event was created is identified on the slave by its server ID. The `ORIGINATOR` column in `INFORMATION_SCHEMA.EVENTS` and the `originator` column in `mysql.event` were added to these tables in MySQL 5.1.18 to

store this information. (See [Section 23.20](#), “The `INFORMATION_SCHEMA` `EVENTS` Table, and [Section 12.5.4.16](#), “`SHOW EVENTS`”.)

- The feature implementation resides on the slave in a renewable state so that if the master fails, the slave can be used as the master without loss of event processing.

To determine whether there are any scheduled events on a MySQL server that were created on a different server (that was acting as a replication master), use `SHOW EVENTS`, like this:

```
SHOW EVENTS
WHERE STATUS = 'SLAVESIDE_DISABLED';
```

Alternatively, you might wish to query the `INFORMATION_SCHEMA.EVENTS` table as shown here:

```
SELECT EVENT_SCHEMA, EVENT_NAME, ORIGINATOR
FROM INFORMATION_SCHEMA.EVENTS
WHERE STATUS = 'SLAVESIDE_DISABLED';
```

When promoting a replication slave having such events to a replication master, use the following query to enable the events:

```
UPDATE mysql.event
SET STATUS = 'ENABLED'
WHERE STATUS = 'SLAVESIDE_DISABLED';
```

If more than one master was involved in creating events on this slave, and you wish to enable events that were created only on a given master having the server ID `master_id`, use the following query instead:

```
UPDATE mysql.event
SET STATUS = 'ENABLED'
WHERE ORIGINATOR = master_id
AND STATUS = 'SLAVESIDE_DISABLED';
```

Important

Before executing either of the previous two `UPDATE` statements, you should disable the Event Scheduler on the slave (using `SET GLOBAL EVENT_SCHEDULER = OFF;`), run the `UPDATE`, restart the server, then re-enable the Event Scheduler afterwards (using `SET GLOBAL EVENT_SCHEDULER = ON;`).

If you later demote the new master back to being a replication slave, you must disable manually all events enabled by the `UPDATE` statement. You can do this by storing in a separate table the event names from the `SELECT` statement shown previously, or using an `UPDATE` statement to rename the events with a common prefix to identify them, as shown in this example:

```
UPDATE mysql.event
SET name = CONCAT('replicated_', name)
WHERE status = 'SLAVESIDE_DISABLED';
```

When demoting this server back to being a replication slave, you can then rename and disable the events like this:

```
UPDATE mysql.event
SET name = REPLACE(name, 'replicated_', ''),
status = 'SLAVESIDE_DISABLED'
WHERE INSTR(name, 'replicated_') = 1;
```

15.3.1.6. Replication with Floating Point Values

Floating-point values are approximate, so comparisons involving them are inexact. This is true for operations that use floating-point values explicitly, or values that are converted to floating-point implicitly. Comparisons of floating-point values might yield different results on master and slave servers due to differences in computer architecture, the compiler used to build MySQL, and so forth. See [Section 11.2.2](#), “Type Conversion in Expression Evaluation”, and [Section B.1.5.8](#), “Problems with Floating-Point Comparisons”.

15.3.1.7. Replication and `FLUSH`

Some forms of the `FLUSH` statement are not logged because they could cause problems if replicated to a slave: `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK`. For a syntax example, see [Section 12.5.5.2](#), “`FLUSH` Syntax”. The `FLUSH TABLES`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements are written to the binary log and thus replicated to slaves. This is not normally a problem because these statements do not modify table data. However, this can cause dif-

difficulties under certain circumstances. If you replicate the privilege tables in the `mysql` database and update those tables directly without using `GRANT`, you must issue a `FLUSH PRIVILEGES` on the slaves to put the new privileges into effect. In addition, if you use `FLUSH TABLES` when renaming a `MyISAM` table that is part of a `MERGE` table, you must issue `FLUSH TABLES` manually on the slaves. These statements are written to the binary log unless you specify `NO_WRITE_TO_BINLOG` or its alias `LOCAL`.

15.3.1.8. Replication and System Functions

Certain functions do not replicate well under some conditions:

- The `USER()`, `CURRENT_USER()`, `UUID()`, `VERSION()`, and `LOAD_FILE()` functions are replicated without change and thus do not work reliably on the slave unless row-based replication is enabled. (See [Section 15.1.2](#), “Replication Formats”.)

For early implementations of mixed-format logging, stored functions, triggers, and views that use these functions in their body do not replicate reliably in mixed-format logging mode because the logging did not switch from statement-based to row-based format. For example, `INSERT INTO t SELECT FROM v`, where `v` is a view that selects `UUID()` could cause problems. This limitation is lifted in MySQL 5.1.12.

- Unlike `NOW()`, the `SYSDATE()` function is not replication-safe because it is not affected by `SET TIMESTAMP` statements in the binary log and is non-deterministic if statement-based logging is used. This is not a problem if row-based logging is used. Another option is to start the server with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`.
- The following restriction applies to statement-based replication only, not to row-based replication.* The `GET_LOCK()`, `RELEASE_LOCK()`, `IS_FREE_LOCK()`, and `IS_USED_LOCK()` functions that handle user-level locks are replicated without the slave knowing the concurrency context on master. Therefore, these functions should not be used to insert into a master's table because the content on the slave would differ. (For example, do not issue a statement such as `INSERT INTO mytable VALUES(GET_LOCK(...))`.)

As a workaround for the preceding limitations when statement-based replication is in effect, you can use the strategy of saving the problematic function result in a user variable and referring to the variable in a later statement. For example, the following single-row `INSERT` is problematic due to the reference to the `UUID()` function:

```
INSERT INTO t VALUES(UUID());
```

To work around the problem, do this instead:

```
SET @my_uuid = UUID();
INSERT INTO t VALUES(@my_uuid);
```

That sequence of statements replicates because the value of `@my_uuid` is stored in the binary log as a user-variable event prior to the `INSERT` statement and is available for use in the `INSERT`.

The same idea applies to multiple-row inserts, but is more cumbersome to use. For a two-row insert, you can do this:

```
SET @my_uuid1 = UUID(); @my_uuid2 = UUID();
INSERT INTO t VALUES(@my_uuid1),(@my_uuid2);
```

However, if the number of rows is large or unknown, the workaround is difficult or impracticable. For example, you cannot convert the following statement to one in which a given individual user variable is associated with each row:

```
INSERT INTO t2 SELECT UUID(), * FROM t1;
```

15.3.1.9. Replication during a Master Crash

A crash on the master side can result in the master's binary log having a final position less than the most recent position read by the slave, due to the master's binary log file not being flushed. This can cause the slave not to be able to replicate when the master comes back up. Setting `sync_binlog=1` in the master `my.cnf` file helps to minimize this problem because it causes the master to flush its binary log more frequently.

15.3.1.10. Replication during a Master Shutdown

It is safe to shut down a master server and restart it later. When a slave loses its connection to the master, the slave tries to reconnect immediately and retries periodically if that fails. The default is to retry every 60 seconds. This may be changed with the -

`-master-connect-retry` option. A slave also is able to deal with network connectivity outages. However, the slave notices the network outage only after receiving no data from the master for `slave_net_timeout` seconds. If your outages are short, you may want to decrease `slave_net_timeout`. See [Section 5.2.3, “System Variables”](#).

15.3.1.11. Replication with **MEMORY** Table Types

When a server shuts down and restarts, its **MEMORY** tables become empty. The master replicates this effect to slaves as follows: The first time that the master uses each **MEMORY** table after startup, it logs an event that notifies the slaves that the table needs to be emptied by writing a **DELETE** statement for that table to the binary log. See [Section 13.7, “The MEMORY \(HEAP\) Storage Engine”](#), for more information about **MEMORY** tables.

15.3.1.12. Replication of the System **mysql** Database

For MySQL 5.1.14 and later, the **mysql** database is not replicated. The **mysql** database is instead seen as a node specific database. Row-based replication is not supported on this table. Instead, statements that would normally update this information (including **GRANT**, **REVOKE**, and the manipulation of triggers, stored routines/procedures, and views) are all replicated to slaves using Statement based replication.

For MySQL 5.1.13 and earlier, user privileges are replicated only if the **mysql** database is replicated. That is, the **GRANT**, **REVOKE**, **SET PASSWORD**, **CREATE USER**, and **DROP USER** statements take effect on the slave only if the replication setup includes the **mysql** database.

If you're replicating all databases, but don't want statements that affect user privileges to be replicated, set up the slave to not replicate the **mysql** database, using the `--replicate-wild-ignore-table=mysql.%` option. The slave will recognize that issuing privilege-related SQL statements won't have an effect, and thus not execute those statements.

15.3.1.13. Slave Errors during Replication

If a statement on a slave produces an error, the slave SQL thread terminates, and the slave writes a message to its error log. You should then connect to the slave manually and determine the cause of the problem. (`SHOW SLAVE STATUS` is useful for this.) Then fix the problem (for example, you might need to create a non-existent table) and run `START SLAVE`.

MySQL Enterprise

For instant notification when a slave thread terminates subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

15.3.1.14. Replication during a Slave Shutdown

Shutting down the slave (cleanly) is also safe because it keeps track of where it left off. Unclean shutdowns might produce problems, especially if the disk cache was not flushed to disk before the system went down. Your system fault tolerance is greatly increased if you have a good uninterruptible power supply. Unclean shutdowns of the master may cause inconsistencies between the content of tables and the binary log in master; this can be avoided by using **InnoDB** tables and the `--innodb-safe-binlog` option on the master. See [Section 5.11.4, “The Binary Log”](#).

15.3.1.15. Replication and Temporary Tables

This item does not apply when row-based replication is in use because in that case temporary tables are not replicated (see [Section 15.1.2, “Replication Formats”](#)).

Temporary tables are replicated except in the case where you shut down the slave server (not just the slave threads) and you have replicated temporary tables that are used in updates that have not yet been executed on the slave. If you shut down the slave server, the temporary tables needed by those updates are no longer available when the slave is restarted. To avoid this problem, do not shut down the slave while it has temporary tables open. Instead, use the following procedure:

1. Issue a `STOP SLAVE` statement.
2. Use `SHOW STATUS` to check the value of the `Slave_open_temp_tables` variable.
3. If the value is 0, issue a `mysqladmin shutdown` command to stop the slave.
4. If the value is not 0, restart the slave threads with `START SLAVE`.
5. Repeat the procedure later until the `Slave_open_temp_tables` variable is 0 and you can stop the slave.

15.3.1.16. Replication Retries and Timeouts

The global system variable `slave_transaction_retries` affects replication as follows: If the replication slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or because it exceeded the `InnoDB innodb_lock_wait_timeout` value, or the `NDBCluster TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout` value, the transaction is automatically retried `slave_transaction_retries` times before stopping with an error. The default value is 10. The total retry count can be seen in the output of `SHOW STATUS`; see [Section 5.2.5, “Status Variables”](#).

15.3.1.17. Replication and Time zones

If the master uses MySQL 4.1, the same system time zone should be set for both master and slave. Otherwise some statements will not be replicated properly, such as statements that use the `NOW()` or `FROM_UNIXTIME()` functions. You can set the time zone in which MySQL server runs by using the `--timezone=timezone_name` option of the `mysqld_safe` script or by setting the `TZ` environment variable. Both master and slave should also have the same default connection time zone setting; that is, the `--default-time-zone` parameter should have the same value for both master and slave. Note that this is not necessary when the master is MySQL 5.0 or later.

`CONVERT_TZ(..., ..., @@session.time_zone)` is properly replicated only if both master and slave are running MySQL 5.0.4 or newer.

15.3.1.18. Replication and Transactions

It is possible to replicate transactional tables on the master using non-transactional tables on the slave. For example, you can replicate an `InnoDB` master table as a `MyISAM` slave table. However, if you do this, there are problems if the slave is stopped in the middle of a `BEGIN/COMMIT` block because the slave restarts at the beginning of the `BEGIN` block.

In situations where transactions mix updates to transactional and non-transactional tables, the order of statements in the binary log is correct, and all needed statements are written to the binary log even in case of a `ROLLBACK`. However, when a second connection updates the non-transactional table before the first connection's transaction is complete, statements can be logged out of order, because the second connection's update is written immediately after it is performed, regardless of the state of the transaction being performed by the first connection.

Due to the non-transactional nature of `MyISAM` tables, it is possible to have a statement that only partially updates a table and returns an error code. This can happen, for example, on a multiple-row insert that has one row violating a key constraint, or if a long update statement is killed after updating some of the rows. If that happens on the master, the slave thread exits and waits for the database administrator to decide what to do about it unless the error code is legitimate and execution of the statement results in the same error code on the slave. If this error code validation behavior is not desirable, some or all errors can be masked out (ignored) with the `--slave-skip-errors` option.

Caution

You should not use transactions in a replication environment that update both transactional and non-transactional tables.

When the storage engine type of the slave is non-transactional, transactions on the master that mix updates of transactional and non-transactional tables should be avoided because they can cause inconsistency of the data between the master's transactional table and the slave's non-transactional table. That is, such transactions can lead to master storage engine-specific behavior with the possible effect of replication going out of synchrony. MySQL does not issue a warning about this currently, so extra care should be taken when replicating transactional tables from the master to non-transactional ones on the slaves.

15.3.1.19. Replication with More Columns on the Slave

Note

Starting with MySQL 5.1.21 you can replicate from master to slave using a different number of columns and (within limits) different table definitions. See [Section 15.3.1.20, “Replication with Fewer Columns on the Slave”](#).

You may replicate from a master to a slave where the number of columns in the table on the slave is larger than the number of columns in the corresponding table on the master.

When replicating tables from the master to tables on a slave with extra columns, the following must be met:

- You must be using row-based replication.

- Master and slave must have the same database/table name.
- Slave tables may contain additional columns compared to the Master, but the columns must appear sequentially after the corresponding columns on the Master.
- All the matching columns on the Master and the Slave must have the same type.
- Every extra column on the slave must have a default value.

This functionality was added in MySQL 5.1.12.

15.3.1.20. Replication with Fewer Columns on the Slave

Starting with MySQL 5.1.21 you can replicate from master to slave with a different number of columns (fewer or greater) on each host.

- You must be using row-based replication.
- Master and slave must have the same database/table name.
- Slave tables may contain additional columns compared to the Master, but the columns must appear sequentially after the corresponding columns on the Master. For example, the following definitions would replicate correctly:

```
mysql> CREATE TABLE master (first INT, second INT);
mysql> CREATE TABLE slave (first INT, second INT, third INT);
```

But these table definitions would raise error **1532 SQLSTATE: HY000 (ER_BINLOG_ROW_RBR_TO_SBR)** because the column definitions on the slave are in a different order than the master:

```
mysql> CREATE TABLE master (first INT, second INT);
mysql> CREATE TABLE slave (third INT, second INT, first INT);
```

- Master tables can have more columns compared to the columns on the slave, but the additional columns on the master must appear after the matching columns in the server. The following table definitions would be valid:

```
mysql> CREATE TABLE master (first INT, second INT, third INT);
mysql> CREATE TABLE slave (first INT, second INT);
```

But these table definitions would raise error 1532 because the column definitions on the slave are in a different order than the master:

```
mysql> CREATE TABLE master (third INT, second INT, first INT);
mysql> CREATE TABLE slave (first INT, second INT);
```

- Columns on the master and slave should have the same type, but within certain limits, this is not strictly enforced:
 - For numeric column types, the types must match. There is no conversion between **INT** and **FLOAT** types.
 - **BINARY**, **VARBINARY**, **CHAR** and **VARCHAR** columns are treated as equal. Therefore you can replicate from **CHAR** to **BINARY**.
 - For **CHAR** and **BINARY** column types, the column size on the slave must be at least equal to the column size on the master. If the column size on the master is greater than on the slave, error 1532 will be raised.
- When replicating with between a master and the slave where the number of columns on the slave is greater than on the master, every additional column on the slave must have a default value.

15.3.1.21. Replication and Variables

The **FOREIGN_KEY_CHECKS**, **SQL_MODE**, **UNIQUE_CHECKS**, and **SQL_AUTO_IS_NULL** variables are all replicated (this has been true since MySQL 5.0). The **storage_engine** system variable (also known as **table_type**) is not yet replicated in MySQL 5.1, which is a good thing for replication between different storage engines.

Session variables are not replicated properly when used in statements that update tables. For example, **SET MAX_JOIN_SIZE=1000**

followed by `INSERT INTO mytable VALUES(@@MAX_JOIN_SIZE)` will not insert the same data on the master and the slave. This does not apply to the common sequence of `SET TIME_ZONE=...` followed by `INSERT INTO mytable VALUES(CONVERT_TZ(..., @@time_zone))`.

Replication of session variables is not a problem when row-based replication is being used. See [Section 15.1.2, “Replication Formats”](#).

15.3.1.22. Replication and Views

Views are always replicated to slaves. Views are filtered by their own name, not by the tables they refer to. This means that a view can be replicated to the slave even if the view contains a table that would normally be filtered out by [replication-ignore-table](#) rules. Care should therefore be taken to ensure that views do not replicate table data that would normally be filtered for security reasons.

15.3.2. Replication Compatibility Between MySQL Versions

The binary log format as implemented in MySQL 5.1 is considerably different from that used in previous versions, especially with regard to handling of character sets, `LOAD DATA INFILE`, and time zones.

As a general rule, you should setup replication only between masters and slaves running the same major versions (5.1, 5.0 or 4.1) of MySQL. If you must execute replication between different major versions, ensure that your client is at a version equal to or higher than that of the master. For example, a master of 4.1.23 and a slave of 5.0.24.

We recommend using the most recent MySQL version available because replication capabilities are continually being improved. We also recommend using the same version for both the master and the slave. We recommend upgrading masters and slaves running alpha or beta versions to new (production) versions. In many cases, replication from a newer master to an older slave will fail. In general, slaves running MySQL 5.1.x can be used with older masters (even those running MySQL 3.23, 4.0, or 4.1), but not the reverse.

Note

You *cannot* replicate from a master that uses a newer binary log format to a slave that uses an older format (for example, from MySQL 5.0 to MySQL 4.1.) This has significant implications for upgrading replication servers, as described in [Section 15.3.3, “Upgrading a Replication Setup”](#).

The preceding information pertains to replication compatibility at the protocol level. However, there can be other constraints, such as SQL-level compatibility issues. For example, a 5.1 master cannot replicate to a 5.0 slave if the replicated statements use SQL features available in 5.1 but not in 5.0. These and other issues are discussed in [Section 15.3.1, “Replication Features and Issues”](#).

15.3.3. Upgrading a Replication Setup

When you upgrade servers that participate in a replication setup, the procedure for upgrading depends on the current server versions and the version to which you are upgrading.

This section applies to upgrading replication from MySQL 3.23, 4.0, or 4.1 to MySQL 5.1. A 4.0 server should be 4.0.3 or newer.

When you upgrade a master to 5.1 from an earlier MySQL release series, you should first ensure that all the slaves of this master are using the same 5.1.x release. If this is not the case, you should first upgrade the slaves. To upgrade each slave, shut it down, upgrade it to the appropriate 5.1.x version, restart it, and restart replication. The 5.1 slave is able to read the old relay logs written prior to the upgrade and to execute the statements they contain. Relay logs created by the slave after the upgrade are in 5.1 format.

After the slaves have been upgraded, shut down the master, upgrade it to the same 5.1.x release as the slaves, and restart it. The 5.1 master is able to read the old binary logs written prior to the upgrade and to send them to the 5.1 slaves. The slaves recognize the old format and handle it properly. Binary logs created by the master following the upgrade are in 5.1 format. These too are recognized by the 5.1 slaves.

In other words, there are no measures to take when upgrading to MySQL 5.1, except that the slaves must be MySQL 5.1 before you can upgrade the master to 5.1. Note that downgrading from 5.1 to older versions does not work so simply: You must ensure that any 5.1 binary logs or relay logs have been fully processed, so that you can remove them before proceeding with the downgrade.

Downgrading a replication setup to a previous version cannot be done once you've switched from statement-based to row-based replication, and after the first row-based statement has been written to the binlog. See [Section 15.1.2, “Replication Formats”](#).

15.3.4. Replication FAQ

MySQL Enterprise

For expert advice on replication subscribe to the MySQL Enterprise Monitor. For more information see [ht-](#)

[tp://www.mysql.com/products/enterprise/advisors.html](http://www.mysql.com/products/enterprise/advisors.html).

Questions

- [16.3.4.1](#): Does the slave need to be connected to the master all the time?
- [16.3.4.2](#): Do I have to enable networking on my master to enable replication?
- [16.3.4.3](#): How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave?
- [16.3.4.4](#): How do I force the master to block updates until the slave catches up?
- [16.3.4.5](#): What issues should I be aware of when setting up two-way replication?
- [16.3.4.6](#): How can I use replication to improve performance of my system?
- [16.3.4.7](#): What should I do to prepare client code in my own applications to use performance-enhancing replication?
- [16.3.4.8](#): When and how much can MySQL replication improve the performance of my system?
- [16.3.4.9](#): How can I use replication to provide redundancy or high availability?
- [16.3.4.10](#): How do I tell whether a master server is using statement-based or row-based binary logging format?
- [16.3.4.11](#): How do I tell a slave to use row-based replication?
- [16.3.4.12](#): How do I prevent GRANT and REVOKE statements from replicating to slave machines?
- [16.3.4.13](#): Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on Mac OS X and Windows)?
- [16.3.4.14](#): Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)?

Questions and Answers

16.3.4.1: Does the slave need to be connected to the master all the time?

No, it does not. The slave can go down or stay disconnected for hours or even days, and then reconnect and catch up on updates. For example, you can set up a master/slave relationship over a dial-up link where the link is up only sporadically and for short periods of time. The implication of this is that, at any given time, the slave is not guaranteed to be in sync with the master unless you take some special measures.

To ensure that this is the case, you must not remove binary logs from the master, where the information has not been replicated to the slaves. Asynchronous replication can only work if the slave is able to read the binary log from the last point in the binary logs where it had read the replication statements.

16.3.4.2: Do I have to enable networking on my master to enable replication?

Networking must be enabled on the master. If networking is not enabled, then the slave is unable to connect to the master and transfer the binary log. Check that the [skip-networking](#) option has not been enabled in your configuration file.

16.3.4.3: How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave?

You can read the [Seconds_Behind_Master](#) column in `SHOW SLAVE STATUS`. See [Section 15.4.1, “Replication Implementation Details”](#).

When the slave SQL thread executes an event read from the master, it modifies its own time to the event timestamp. (This is why [TIMESTAMP](#) is well replicated.) In the [Time](#) column in the output of `SHOW PROCESSLIST`, the number of seconds displayed for the slave SQL thread is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. You can use this to determine the date of the last replicated event. Note that if your slave has been disconnected from the master for one hour, and then reconnects, you may immediately see [Time](#) values like 3600 for the slave SQL thread in `SHOW PROCESSLIST`. This is because the slave is executing statements that are one hour old.

16.3.4.4: How do I force the master to block updates until the slave catches up?

Use the following procedure:

1. On the master, execute these statements:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

Record the replication coordinates (the log filename and offset) from the output of the `SHOW` statement.

2. On the slave, issue the following statement, where the arguments to the `MASTER_POS_WAIT()` function are the replication coordinate values obtained in the previous step:

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_offset);
```

The `SELECT` statement blocks until the slave reaches the specified log file and offset. At that point, the slave is in synchrony with the master and the statement returns.

3. On the master, issue the following statement to allow the master to begin processing updates again:

```
mysql> UNLOCK TABLES;
```

16.3.4.5: What issues should I be aware of when setting up two-way replication?

MySQL replication currently does not support any locking protocol between master and slave to guarantee the atomicity of a distributed (cross-server) update. In other words, it is possible for client A to make an update to co-master 1, and in the meantime, before it propagates to co-master 2, client B could make an update to co-master 2 that makes the update of client A work differently than it did on co-master 1. Thus, when the update of client A makes it to co-master 2, it produces tables that are different from what you have on co-master 1, even after all the updates from co-master 2 have also propagated. This means that you should not chain two servers together in a two-way replication relationship unless you are sure that your updates can safely happen in any order, or unless you take care of mis-ordered updates somehow in the client code.

You should also realize that two-way replication actually does not improve performance very much (if at all) as far as updates are concerned. Each server must do the same number of updates, just as you would have a single server do. The only difference is that there is a little less lock contention, because the updates originating on another server are serialized in one slave thread. Even this benefit might be offset by network delays.

16.3.4.6: How can I use replication to improve performance of my system?

You should set up one server as the master and direct all writes to it. Then configure as many slaves as you have the budget and rack-space for, and distribute the reads among the master and the slaves. You can also start the slaves with the `--skip-innodb`, `--low-priority-updates`, and `--delay-key-write=ALL` options to get speed improvements on the slave end. In this case, the slave uses non-transactional `MyISAM` tables instead of `InnoDB` tables to get more speed by eliminating transactional overhead.

16.3.4.7: What should I do to prepare client code in my own applications to use performance-enhancing replication?

See the guide to using replication as a scale-out solution, [Section 15.2.3, “Using Replication for Scale-out”](#).

16.3.4.8: When and how much can MySQL replication improve the performance of my system?

MySQL replication is most beneficial for a system that processes frequent reads and infrequent writes. In theory, by using a single-master/multiple-slave setup, you can scale the system by adding more slaves until you either run out of network bandwidth, or your update load grows to the point that the master cannot handle it.

To determine how many slaves you can use before the added benefits begin to level out, and how much you can improve performance of your site, you need to know your query patterns, and to determine empirically by benchmarking the relationship between the throughput for reads (reads per second, or `reads`) and for writes (`writes`) on a typical master and a typical slave. The example here shows a rather simplified calculation of what you can get with replication for a hypothetical system.

Let's say that system load consists of 10% writes and 90% reads, and we have determined by benchmarking that `reads` is $1200 - 2 \times \text{writes}$. In other words, the system can do 1,200 reads per second with no writes, the average write is twice as slow as the average read, and the relationship is linear. Let us suppose that the master and each slave have the same capacity, and that we have one master

and N slaves. Then we have for each server (master or slave):

`reads = 1200 - 2 × writes`

`reads = 9 × writes / (N + 1)` (reads are split, but writes go to all servers)

`9 × writes / (N + 1) + 2 × writes = 1200`

`writes = 1200 / (2 + 9/(N+1))`

The last equation indicates the maximum number of writes for N slaves, given a maximum possible read rate of 1,200 per minute and a ratio of nine reads per write.

This analysis yields the following conclusions:

- If $N = 0$ (which means we have no replication), our system can handle about $1200/11 = 109$ writes per second.
- If $N = 1$, we get up to 184 writes per second.
- If $N = 8$, we get up to 400 writes per second.
- If $N = 17$, we get up to 480 writes per second.
- Eventually, as N approaches infinity (and our budget negative infinity), we can get very close to 600 writes per second, increasing system throughput about 5.5 times. However, with only eight servers, we increase it nearly four times.

Note that these computations assume infinite network bandwidth and neglect several other factors that could be significant on your system. In many cases, you may not be able to perform a computation similar to the one just shown that accurately predicts what will happen on your system if you add N replication slaves. However, answering the following questions should help you decide whether and by how much replication will improve the performance of your system:

- What is the read/write ratio on your system?
- How much more write load can one server handle if you reduce the reads?
- For how many slaves do you have bandwidth available on your network?

16.3.4.9: How can I use replication to provide redundancy or high availability?

How you implement redundancy is entirely dependent on your application and circumstances. High-availability solutions (with automatic failover) require active monitoring and either custom scripts or third party tools to provide the failover support from the original MySQL server to the slave.

To handle the process manually, you should be able to switch from a failed master to a pre-configured slave by altering your application to talk to the new server or by adjusting the DNS for the MySQL server from the failed server to the new server.

For more information and some example solutions, see [Section 15.2.6, “Switching Masters During Failover”](#).

16.3.4.10: How do I tell whether a master server is using statement-based or row-based binary logging format?

Check the value of the `binlog_format` system variable:

```
mysql> SHOW VARIABLES LIKE 'binlog_format';
```

The value will be either `STATEMENT` or `ROW`.

16.3.4.11: How do I tell a slave to use row-based replication?

Slaves automatically know which format to use.

16.3.4.12: How do I prevent GRANT and REVOKE statements from replicating to slave machines?

Start the server with the `--replicate-wild-ignore-table=mysql.%` option.

16.3.4.13: Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on Mac OS X and Windows)?

Yes.

16.3.4.14: Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)?

Yes.

15.3.5. Troubleshooting Replication

If you have followed the instructions, and your replication setup is not working, the first thing to do is *check the error log for messages*. Many users have lost time by not doing this soon enough after encountering problems.

If you cannot tell from the error log what the problem was, try the following techniques:

- Verify that the master has binary logging enabled by issuing a `SHOW MASTER STATUS` statement. If logging is enabled, `Position` is non-zero. If binary logging is not enabled, verify that you are running the master with the `--log-bin` and `--server-id` options.
- Verify that the slave is running. Use `SHOW SLAVE STATUS` to check whether the `Slave_IO_Running` and `Slave_SQL_Running` values are both `Yes`. If not, verify the options that were used when starting the slave server. For example, `--skip-slave-start` prevents the slave threads from starting until you issue a `START SLAVE` statement.
- If the slave is running, check whether it established a connection to the master. Use `SHOW PROCESSLIST`, find the I/O and SQL threads and check their `State` column to see what they display. See [Section 15.4.1, “Replication Implementation Details”](#). If the I/O thread state says `Connecting to master`, check the following:
 - Verify the privileges for the user being used for replication on the master.
 - Check that the hostname of the master is correct and that you are using the correct port to connect to the master. The port used for replication is the same as used for client network communication (the default is `3306`). For the hostname, ensure that the name resolves to the correct IP address.
 - Check that networking on the master and slave have not been disabled. Look for the `skip-networking` option in the configuration file. It should either be commented out or deleted entirely.
 - If the master has a firewall or IP filtering configuration, ensure that the network port being used for MySQL is not being filtered.
 - Check that you can reach the master by using `ping` or `traceroute/tracert` to reach the host.
- If the slave was running previously but has stopped, the reason usually is that some statement that succeeded on the master failed on the slave. This should never happen if you have taken a proper snapshot of the master, and never modified the data on the slave outside of the slave thread. If the slave stops unexpectedly, it is a bug or you have encountered one of the known replication limitations described in [Section 15.3.1, “Replication Features and Issues”](#). If it is a bug, see [Section 15.3.6, “How to Report Replication Bugs or Problems”](#), for instructions on how to report it.
- If a statement that succeeded on the master refuses to run on the slave, try the following procedure if it is not feasible to do a full database resynchronization by deleting the slave's databases and copying a new snapshot from the master:
 1. Determine whether the affected table on the slave is different from the master table. Try to understand how this happened. Then make the slave's table identical to the master's and run `START SLAVE`.
 2. If the preceding step does not work or does not apply, try to understand whether it would be safe to make the update manually (if needed) and then ignore the next statement from the master.
 3. If you decide that you can skip the next statement from the master, issue the following statements:

```
mysql> SET GLOBAL SQL_SLAVE_SKIP_COUNTER = N;  
mysql> START SLAVE;
```

The value of `N` should be 1 if the next statement from the master does not use `AUTO_INCREMENT` or `LAST_INSERT_ID()`. Otherwise, the value should be 2. The reason for using a value of 2 for statements that use `AUTO_INCREMENT` or

`LAST_INSERT_ID()` is that they take two events in the binary log of the master.

4. If you are sure that the slave started out perfectly synchronized with the master, and that no one has updated the tables involved outside of the slave thread, then presumably the discrepancy is the result of a bug. If you are running the most recent version of MySQL, please report the problem. If you are running an older version, try upgrading to the latest production release to determine whether the problem persists.

15.3.6. How to Report Replication Bugs or Problems

When you have determined that there is no user error involved, and replication still either does not work at all or is unstable, it is time to send us a bug report. We need to obtain as much information as possible from you to be able to track down the bug. Please spend some time and effort in preparing a good bug report.

If you have a repeatable test case that demonstrates the bug, please enter it into our bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#). If you have a “phantom” problem (one that you cannot duplicate at will), use the following procedure:

1. Verify that no user error is involved. For example, if you update the slave outside of the slave thread, the data goes out of synchrony, and you can have unique key violations on updates. In this case, the slave thread stops and waits for you to clean up the tables manually to bring them into synchrony. *This is not a replication problem. It is a problem of outside interference causing replication to fail.*
2. Run the slave with the `--log-slave-updates` and `--log-bin` options. These options cause the slave to log the updates that it receives from the master into its own binary logs.
3. Save all evidence before resetting the replication state. If we have no information or only sketchy information, it becomes difficult or impossible for us to track down the problem. The evidence you should collect is:
 - All binary logs from the master
 - All binary logs from the slave
 - The output of `SHOW MASTER STATUS` from the master at the time you discovered the problem
 - The output of `SHOW SLAVE STATUS` from the slave at the time you discovered the problem
 - Error logs from the master and the slave
4. Use `mysqlbinlog` to examine the binary logs. The following should be helpful to find the problem statement. `log_pos` and `log_file` are the `Master_Log_File` and `Read_Master_Log_Pos` values from `SHOW SLAVE STATUS`.

```
shell> mysqlbinlog -j log_pos log_file | head
```

After you have collected the evidence for the problem, try to isolate it as a separate test case first. Then enter the problem with as much information as possible into our bugs database using the instructions at [Section 1.8, “How to Report Bugs or Problems”](#).

15.4. Replication Implementation

The basic mechanics of replication is based on the master server keeping track of all changes to your databases (updates, deletes, and so on) in its binary logs. The binary log serves as a written record of each to the database from the moment the database was started. The binary log contains records of all the statements which edit or modify either the database structure or the data that the structure contains. Typically `SELECT` statements are not recorded, as they do not modify the database data or structure.

Each slave that connects to the master receives a copy of the binary log, and executes the events within the binary log. This has the effect of repeating the original statements and changes just as they were made on the master. Tables are created or their structure modified, and data is inserted, deleted and updated according to the statements that were originally executed on the master.

Because each slave is independent, the replaying of the statements in the masters binary log can occur on each slave that is connected to the master. In addition, because each slave only receives a copy of the binary log by requesting it from the master (it pulls the data from the master, rather than the master pushing the data to the slave), the slave is able to read and update the copy of the database at it's own

pace and rate and can start and stop the replication process at will without affecting the master or the slaves ability to update to the latest database status.

For more information on the specifics of the replication implementation, see [Section 15.4.1, “Replication Implementation Details”](#).

Slaves and masters report their status in respect of the replication process regularly so that you can monitor the situation. For information on slave states, see [Section 6.5.5.5, “Replication Slave I/O Thread States”](#), and [Section 6.5.5.6, “Replication Slave SQL Thread States”](#). For master states, see [Section 6.5.5.4, “Replication Master Thread States”](#).

The master binary log is written to a local relay log on the slave before it is processed. The slave also records information about the current position with the master's binary log and the local relayed log. See [Section 15.4.2, “Replication Relay and Status Files”](#).

Databases and tables are updated on the slave according to a set of rules that are applied according to the various configuration options and variables that control statement evaluation. For details on how these rules are applied, see [Section 15.4.3, “How Servers Evaluate Replication Rules”](#).

15.4.1. Replication Implementation Details

MySQL replication capabilities are implemented using three threads (one on the master server and two on the slave). When a [START SLAVE](#) statement is issued on a slave server, the slave creates an I/O thread, which connects to the master and asks it to send the updates recorded in its binary logs. The master creates a thread to send the binary log contents to the slave. This thread can be identified as the [Binlog Dump](#) thread in the output of [SHOW PROCESSLIST](#) on the master. The slave I/O thread reads the updates that the master [Binlog Dump](#) thread sends and copies them to local files, known as *relay logs*, in the slave's data directory. The third thread is the SQL thread, which the slave creates to read the relay logs and to execute the updates they contain.

MySQL Enterprise

For constant monitoring of the status of slaves subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

In the preceding description, there are three threads per master/slave connection. A master that has multiple slaves creates one thread for each currently-connected slave, and each slave has its own I/O and SQL threads.

The slave uses two threads so that reading updates from the master and executing them can be separated into two independent tasks. Thus, the task of reading statements is not slowed down if statement execution is slow. For example, if the slave server has not been running for a while, its I/O thread can quickly fetch all the binary log contents from the master when the slave starts, even if the SQL thread lags far behind. If the slave stops before the SQL thread has executed all the fetched statements, the I/O thread has at least fetched everything so that a safe copy of the statements is stored locally in the slave's relay logs, ready for execution the next time that the slave starts. This enables the master server to purge its binary logs sooner because it no longer needs to wait for the slave to fetch their contents.

The [SHOW PROCESSLIST](#) statement provides information that tells you what is happening on the master and on the slave regarding replication. See [Section 6.5.5, “Examining Thread Information”](#), for descriptions of all replicated-related states.

The following example illustrates how the three threads show up in the output from [SHOW PROCESSLIST](#).

On the master server, the output from [SHOW PROCESSLIST](#) looks like this:

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
  Id: 2
  User: root
  Host: localhost:32931
  db: NULL
  Command: Binlog Dump
  Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
        be updated
  Info: NULL
```

Here, thread 2 is a [Binlog Dump](#) replication thread for a connected slave. The [State](#) information indicates that all outstanding updates have been sent to the slave and that the master is waiting for more updates to occur. If you see no [Binlog Dump](#) threads on a master server, this means that replication is not running — that is, that no slaves are currently connected.

On the slave server, the output from [SHOW PROCESSLIST](#) looks like this:

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
  Id: 10
```



```

User: system user
Host:
db: NULL
Command: Connect
Time: 11
State: Waiting for master to send event
Info: NULL
***** 2. row *****
Id: 11
User: system user
Host:
db: NULL
Command: Connect
Time: 11
State: Has read all relay log; waiting for the slave I/O
      thread to update it
Info: NULL

```

This information indicates that thread 10 is the I/O thread that is communicating with the master server, and thread 11 is the SQL thread that is processing the updates stored in the relay logs. At the time that the `SHOW PROCESSLIST` was run, both threads were idle, waiting for further updates.

The value in the `Time` column can show how late the slave is compared to the master. See [Section 15.3.4, “Replication FAQ”](#).

15.4.2. Replication Relay and Status Files

During replication the MySQL server creates a number of files that are used to hold the relayed binary log from the master, and record information about the current status and location within the relayed log. There are three file types used in the process:

- The *relay log* consists of the events read from the binary log of the master. Events in this binary log are executed on the slave as part of the replication thread.
- The *master.info* file contains the status and current configuration information for the slave's connectivity to the master. The file holds information on the master hostname, login credentials, and the current position within the master's binary log.
- The *relay.info* file holds the status information about the execution point within the slave's relay log files.

The relationship between the three files and the replication process is as follows. The `master.info` file retains the point within the master binary log that has been read from the master. These read events are written to the relay log. The `relay.info` file records the position within the relay log of the statements that have been executed.

15.4.2.1. The Slave Relay Log

By default, relay logs filenames have the form `host_name-relay-bin.nnnnnn`, where `host_name` is the name of the slave server host and `nnnnnn` is a sequence number. Successive relay log files are created using successive sequence numbers, beginning with `000001`. The slave uses an index file to track the relay log files currently in use. The default relay log index filename is `host_name-relay-bin.index`.

By default, the slave server creates relay log files in its data directory. The default filenames can be overridden with the `--relay-log` and `--relay-log-index` server options. See [Section 15.1.3, “Replication Options and Variables”](#).

Relay logs have the same format as binary logs and can be read using `mysqlbinlog`. The SQL thread automatically deletes each relay log file as soon as it has executed all events in the file and no longer needs it. There is no explicit mechanism for deleting relay logs because the SQL thread takes care of doing so. However, `FLUSH LOGS` rotates relay logs, which influences when the SQL thread deletes them.

A slave server creates a new relay log file under the following conditions:

- Each time the I/O thread starts.
- When the logs are flushed; for example, with `FLUSH LOGS` or `mysqladmin flush-logs`.
- When the size of the current relay log file becomes too large. The meaning of “too large” is determined as follows:
 - If the value of `max_relay_log_size` is greater than 0, that is the maximum relay log file size.
 - If the value of `max_relay_log_size` is 0, `max_binlog_size` determines the maximum relay log file size.

15.4.2.2. The Slave Status Files

A slave replication server creates two small files in the data directory. These *status files* are named `master.info` and `relay-log.info` by default. Their names can be changed by using the `--master-info-file` and `--relay-log-info-file` options. See [Section 15.1.3, “Replication Options and Variables”](#).

The two status files contain information like that shown in the output of the `SHOW SLAVE STATUS` statement, which is discussed in [Section 12.6.2, “SQL Statements for Controlling Slave Servers”](#). Because the status files are stored on disk, they survive a slave server's shutdown. The next time the slave starts up, it reads the two files to determine how far it has proceeded in reading binary logs from the master and in processing its own relay logs.

The I/O thread updates the `master.info` file. The following table shows the correspondence between the lines in the file and the columns displayed by `SHOW SLAVE STATUS`.

Line	Status Column	Description
1		Number of lines in the file
2	<code>Master_Log_File</code>	The name of the master binary log currently being read from the master.
3	<code>Read_Master_Log_Pos</code>	The current position within the master binary log that have been read from the master.
4	<code>Master_Host</code>	The hostname of the master.
5	<code>Master_User</code>	The username used to connect to the master.
6	Password (not shown by <code>SHOW SLAVE STATUS</code>)	The password used to connect to the master.
7	<code>Master_Port</code>	The network port used to connect to the master.
8	<code>Connect_Retry</code>	The period (in seconds) that the slave will wait before trying to re-connect to the master.
9	<code>Master_SSL_Allowed</code>	Indicates whether the server supports SSL connections.
10	<code>Master_SSL_CA_File</code>	The file used for the Certificate Authority (CA) certificate.
11	<code>Master_SSL_CA_Path</code>	The path to the Certificate Authority (CA) certificates.
12	<code>Master_SSL_Cert</code>	The name of the SSL certificate file.
13	<code>Master_SSL_Cipher</code>	The name of the cipher in use for the SSL connection.
14	<code>Master_SSL_Key</code>	The name of the SSL key file.
15	<code>Master_SSL_Verify_Server_Cert</code>	Whether to verify the server certificate.

`Master_SSL_Verify_Server_Cert` is present in `master.info` as of MySQL 5.1.18. It is used as described for the `--ssl-verify-server-cert` option in [Section 5.8.7.3, “SSL Command Options”](#).

The SQL thread updates the `relay-log.info` file. The following table shows the correspondence between the lines in the file and the columns displayed by `SHOW SLAVE STATUS`.

Line	Status Column	Description
1	<code>Relay_Log_File</code>	The name of the current relay log file.
2	<code>Relay_Log_Pos</code>	The current position within the relay log file. Events up to this position have been executed on the slave database.
3	<code>Relay_Master_Log_File</code>	The name of the master binary log file from which the events in the relay log file were read.
4	<code>Exec_Master_Log_Pos</code>	The equivalent position within the master's binary log file of events that have already been executed.

The contents of the `relay-log.info` file and the states shown by the `SHOW SLAVE STATES` command may not match if the `relay-log.info` file has not been flushed to disk. Ideally, you should only view `relay-log.info` on a slave that is offline (i.e. `mysqld` is not running). For a running system, `SHOW SLAVE STATUS` should be used.

15.4.3. How Servers Evaluate Replication Rules

If a master server does not write a statement to its binary log, the statement is not replicated. If the server does log the statement, the statement is sent to all slaves and each slave determines whether to execute it or ignore it.

On the master you can control which databases write events to the binary log using the `--binlog-do-db` and `--binlog-ignore-db` options to control binary logging. For a description of the rules that servers use in evaluating these options, see [Section 5.11.4, “The Binary Log”](#). You should not use these options to control the databases and tables that are replicated, instead, use filtering on the slave to control the events that are executed on the slave.

On the slave side, decisions about whether to execute or ignore statements received from the master are made according to the `--replicate-*` options that the slave was started with. (See [Section 15.1.3, “Replication Options and Variables”](#).) The slave evaluates these options using the following procedure, which first checks the database-level options and then the table-level options.

In the simplest case, when there are no `--replicate-*` options, the procedure yields the result that the slave executes all statements that it receives from the master. Otherwise, the result depends on the particular options given. In general, to make it easier to determine what effect an option set will have, it is recommended that you avoid mixing “do” and “ignore” options, or wildcard and non-wildcard options.

Stage 1. Check the database options.

At this stage, the slave checks whether there are any `--replicate-do-db` or `--replicate-ignore-db` options that specify database-specific conditions:

- *No*: Permit the statement and proceed to the table-checking stage.
- *Yes*: Test the options using the same rules as for the `--binlog-do-db` and `--binlog-ignore-db` options to determine whether to permit or ignore the statement. What is the result of the test?
 - *Permit*: Do not execute the statement immediately. Defer the decision and proceed to the table-checking stage.
 - *Ignore*: Ignore the statement and exit.

This stage can permit a statement for further option-checking, or cause it to be ignored. However, statements that are permitted at this stage are not actually executed yet. Instead, they pass to the following stage that checks the table options.

Stage 2. Check the table options.

First, as a preliminary condition, the slave checks whether statement-based replication is enabled. If so and the statement occurs within a stored function, execute the statement and exit. (If row-based replication is enabled, the slave does not know whether a statement occurred within a stored function on the master, so this condition does not apply.)

Next, the slave checks for table options and evaluates them. If the server reaches this point, it executes all statements if there are no table options. If there are “do” table options, the statement must match one of them if it is to be executed; otherwise, it is ignored. If there are any “ignore” options, all statements are executed except those that match any “ignore” option. The following steps describe how this evaluation occurs in more detail.

1. Are there any `--replicate-*-table` options?

- *No*: There are no table restrictions, so all statements match. Execute the statement and exit.
- *Yes*: There are table restrictions. Evaluate the tables to be updated against them. There might be multiple tables to update, so loop through the following steps for each table looking for a matching option. In this case, the behavior depends on whether statement-based replication or row-based replication is enabled:
 - *Statement-based replication*: Proceed to the next step and begin evaluating the table options in the order shown (first the non-wild options, and then the wild options). Only tables that are to be updated are compared to the options. For example, if the statement is `INSERT INTO sales SELECT * FROM prices`, only `sales` is compared to the options). If several tables are to be updated (multiple-table statement), the first table that matches “do” or “ignore” wins. That is, the server checks the first table against the options. If no decision could be made, it checks the second table against the options, and so on.
 - *Row-based replication*: All table row changes are filtered individually. For multiple-table updates, each table is filtered sep-

arately according to the options. Some updates may be executed and some not, depending on the options and the changes to be made. Row-based replication correctly handles cases that would not replicate correctly with statement-based replication, as in this example which assumes that tables in the `foo` database should be replicated:

```
mysql> USE bar;  
mysql> INSERT INTO foo.sometable VALUES (1);
```

2. Are there any `--replicate-do-table` options?
 - *No*: Proceed to the next step.
 - *Yes*: Does the table match any of them?
 - *No*: Proceed to the next step.
 - *Yes*: Execute the statement and exit.
3. Are there any `--replicate-ignore-table` options?
 - *No*: Proceed to the next step.
 - *Yes*: Does the table match any of them?
 - *No*: Proceed to the next step.
 - *Yes*: Ignore the statement and exit.
4. Are there any `--replicate-wild-do-table` options?
 - *No*: Proceed to the next step.
 - *Yes*: Does the table match any of them?
 - *No*: Proceed to the next step.
 - *Yes*: Execute the statement and exit.
5. Are there any `--replicate-wild-ignore-table` options?
 - *No*: Proceed to the next step.
 - *Yes*: Does the table match any of them?
 - *No*: Proceed to the next step.
 - *Yes*: Ignore the statement and exit.
6. No `--replicate-*-table` option was matched. Is there another table to test against these options?
 - *No*: We have now tested all tables to be updated and could not match any option. Are there `--replicate-do-table` or `--replicate-wild-do-table` options?
 - *No*: There were no “do” table options, so no explicit “do” match is required. Execute the statement and exit.
 - *Yes*: There were “do” table options, so the statement is executed only with an explicit match to one of them. Ignore the statement and exit.
 - *Yes*: Loop.

Examples:

- No `--replicate-*` options at all

The slave executes all statements that it receives from the master.

- `--replicate-*-db` options, but no table options

The slave permits or ignores statements using the database options. Then it executes all statements permitted by those options because there are no table restrictions.

- `--replicate-*-table` options, but no database options

All statements are permitted at the database-checking stage because there are no database conditions. The slave executes or ignores statements based on the table options.

- A mix of database and table options

The slave permits or ignores statements using the database options. Then it evaluates all statements permitted by those options according to the table options. In some cases, this process can yield what might seem a counterintuitive result. Consider the following set of options:

```
[mysqld]  
replicate-do-db      = db1  
replicate-do-table   = db2.mytbl2
```

Suppose that `db1` is the default database and the slave receives this statement:

```
INSERT INTO mytbl1 VALUES(1,2,3);
```

The database is `db1`, which matches the `--replicate-do-db` option at the database-checking stage. The algorithm then proceeds to the table-checking stage. If there were no table options, the statement would be executed. However, because the options include a “do” table option, the statement must match if it is to be executed. The statement does not match, so it is ignored. (The same would happen for any table in `db1`.)

Chapter 16. MySQL Cluster

MySQL Cluster is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. It uses the **NDB Cluster** storage engine to enable running several MySQL servers in a cluster. This storage engine is available in MySQL 5.1 binary releases and in RPMs compatible with most modern Linux distributions.

Note

Binary releases and RPMs are not available for MySQL Cluster 5.1 Carrier Grade Edition, which must be built from source.

MySQL Cluster is currently available and supported on a number of platforms, including Linux, Solaris, Mac OS X, HP-UX, and other Unix-style operating systems on a variety of hardware. For exact levels of support available for on specific combinations of operating system versions, operating system distributions, and hardware platforms, please refer to the [Cluster Supported Platforms list](#) maintained by the MySQL Support Team on the MySQL AB Web site.

MySQL Cluster is *not* currently supported on Microsoft Windows. We are working to make Cluster available on all operating systems supported by MySQL, including Windows, and will update the information provided here as this work continues.

This chapter represents a work in progress, and its contents are subject to revision as MySQL Cluster continues to evolve. Additional information regarding MySQL Cluster can be found on the MySQL AB Web site at <http://www.mysql.com/products/cluster/>.

Information about MySQL MySQL Cluster 5.1 Carrier Grade Edition can be found in [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#) as well as on the MySQL AB web site at <http://www.mysql.com/products/database/clustercge/>.

Additional resources. More information may be found in the following places:

- Answers to some commonly asked questions about Cluster may be found in the [Section A.10, “MySQL 5.1 FAQ — MySQL Cluster”](#).
- The MySQL Cluster mailing list: <http://lists.mysql.com/cluster>.
- The MySQL Cluster Forum: <http://forums.mysql.com/list.php?25>.
- Many MySQL Cluster users and some of the MySQL Cluster developers blog about their experiences with Cluster, and make feeds of these available through [PlanetMySQL](#).
- If you are new to MySQL Cluster, you may find our Developer Zone article [How to set up a MySQL Cluster for two servers](#) to be helpful.

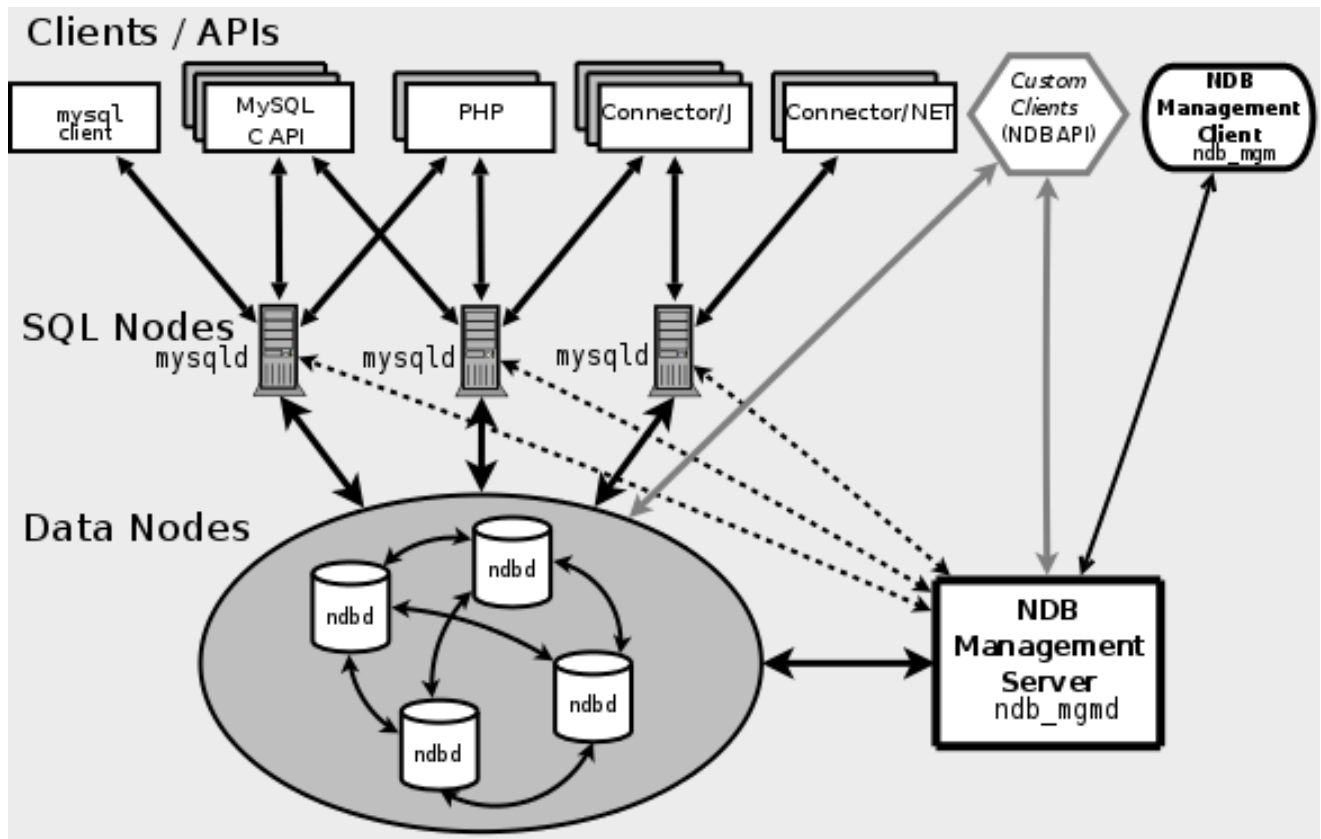
16.1. MySQL Cluster Overview

MySQL Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. The shared-nothing architecture allows the system to work with very inexpensive hardware, and with a minimum of specific requirements for hardware or software.

MySQL Cluster is designed not to have any single point of failure. For this reason, each component is expected to have its own memory and disk, and the use of shared storage mechanisms such as network shares, network filesystems, and SANs is not recommended or supported.

MySQL Cluster integrates the standard MySQL server with an in-memory clustered storage engine called **NDB**. In our documentation, the term **NDB** refers to the part of the setup that is specific to the storage engine, whereas “MySQL Cluster” refers to the combination of MySQL and the **NDB** storage engine.

A MySQL Cluster consists of a set of computers, each running a one or more processes which may include a MySQL server, a data node, a management server, and (possibly) a specialized data access programs. The relationship of these components in a cluster is shown here:



All these programs work together to form a MySQL Cluster. When data is stored in the **NDB Cluster** storage engine, the tables are stored in the data nodes. Such tables are directly accessible from all other MySQL servers in the cluster. Thus, in a payroll application storing data in a cluster, if one application updates the salary of an employee, all other MySQL servers that query this data can see this change immediately.

The data stored in the data nodes for MySQL Cluster can be mirrored; the cluster can handle failures of individual data nodes with no other impact than that a small number of transactions are aborted due to losing the transaction state. Because transactional applications are expected to handle transaction failure, this should not be a source of problems.

16.1.1. MySQL Cluster Core Concepts

NDB is an in-memory storage engine offering high-availability and data-persistence features.

The **NDB** storage engine can be configured with a range of failover and load-balancing options, but it is easiest to start with the storage engine at the cluster level. MySQL Cluster's **NDB** storage engine contains a complete set of data, dependent only on other data within the cluster itself.

The cluster portion of MySQL Cluster is currently configured independently of the MySQL servers. In a MySQL Cluster, each part of the cluster is considered to be a *node*.

Note

In many contexts, the term “node” is used to indicate a computer, but when discussing MySQL Cluster it means a *process*. It is possible to run any number of nodes on a single computer, for which we use the term *cluster host*.

(However, it should be noted MySQL does not currently support the use of multiple data nodes on a single computer in a production setting. See [Section 16.13.10, “Limitations Relating to Multiple Cluster Nodes”](#).)

There are three types of cluster nodes, and in a minimal MySQL Cluster configuration, there will be at least three nodes, one of each of these types:

- **Management node** (MGM node): The role of this type of node is to manage the other nodes within the MySQL Cluster, performing such functions as providing configuration data, starting and stopping nodes, running backup, and so forth. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node. An MGM node is started with the command `ndb_mgmd`.
- **Data node**: This type of node stores cluster data. There are as many data nodes as there are replicas, times the number of fragments. For example, with two replicas, each having two fragments, you will need four data nodes. It is not necessary to have more than one replica. A data node is started with the command `ndbd`.
- **SQL node**: This is a node that accesses the cluster data. In the case of MySQL Cluster, an SQL node is a traditional MySQL server that uses the **NDB Cluster** storage engine. An SQL node is typically started with the command `mysqld --ndbcluster` or by using `mysqld` with the `ndbcluster` option added to `my.cnf`.

An SQL node is actually just a specialised type of *API node*, which designates any application which accesses Cluster data. One example of an API node is the `ndb_restore` utility that is used to restore a cluster backup. It is possible to write such applications using the **NDB API**.

Important

It is not realistic to expect to employ a three-node setup in a production environment. Such a configuration provides no redundancy; in order to benefit from MySQL Cluster's high-availability features, you must use multiple data and SQL nodes. The use of multiple management nodes is also highly recommended.

For a brief introduction to the relationships between nodes, node groups, replicas, and partitions in MySQL Cluster, see [Section 16.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#).

Configuration of a cluster involves configuring each individual node in the cluster and setting up individual communication links between nodes. MySQL Cluster is currently designed with the intention that data nodes are homogeneous in terms of processor power, memory space, and bandwidth. In addition, to provide a single point of configuration, all configuration data for the cluster as a whole is located in one configuration file.

The management server (MGM node) manages the cluster configuration file and the cluster log. Each node in the cluster retrieves the configuration data from the management server, and so requires a way to determine where the management server resides. When interesting events occur in the data nodes, the nodes transfer information about these events to the management server, which then writes the information to the cluster log.

In addition, there can be any number of cluster client processes or applications. These are of two types:

- **Standard MySQL clients.** These are no different for MySQL Cluster than they are for standard (non-Cluster) MySQL. In other words, MySQL Cluster can be accessed from existing MySQL applications written in PHP, Perl, C, C++, Java, Python, Ruby, and so on.
- **Management clients.** These clients connect to the management server and provide commands for starting and stopping nodes gracefully, starting and stopping message tracing (debug versions only), showing node versions and status, starting and stopping backups, and so on.

16.1.2. MySQL Cluster Nodes, Node Groups, Replicas, and Partitions

This section discusses the manner in which MySQL Cluster divides and duplicates data for storage.

Central to an understanding of this topic are the following concepts, listed here with brief definitions:

- **(Data) Node.** An `ndbd` process, which stores a *replica* —that is, a copy of the *partition* (see below) assigned to the node group of which the node is a member.

Each data node should be located on a separate computer. While it is also possible to host multiple `ndbd` processes on a single computer, such a configuration is not supported.

It is common for the terms “node” and “data node” to be used interchangeably when referring to an `ndbd` process; where mentioned, management (MGM) nodes (`ndb_mgmd` processes) and SQL nodes (`mysqld` processes) are specified as such in this discussion.

- **Node Group.** A node group consists of one or more nodes, and stores partitions, or sets of *replicas* (see next item).

Note

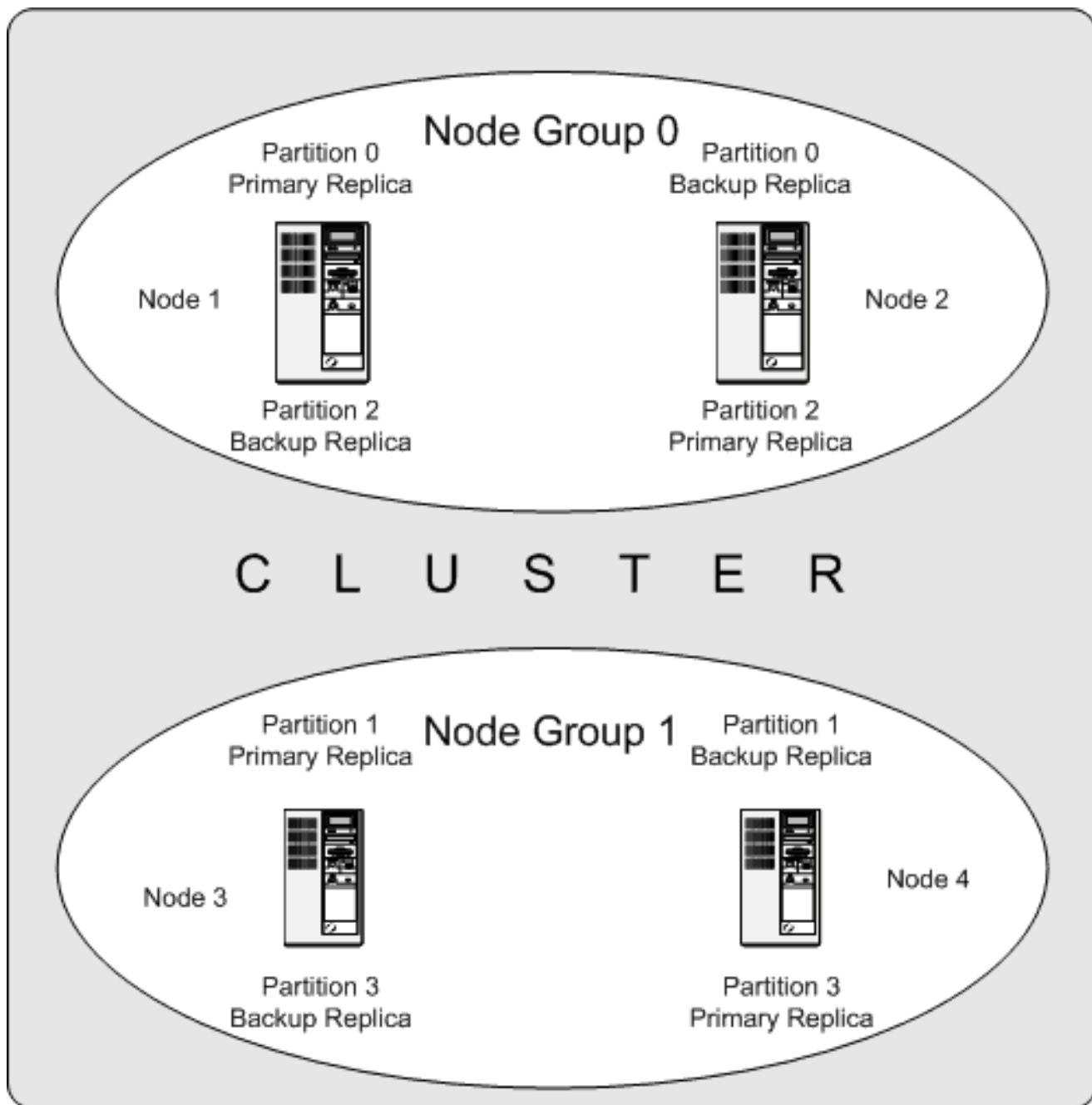
All node groups in a cluster must have the same number of nodes.

- **Partition.** This is a portion of the data stored by the cluster. There are as many cluster partitions as nodes participating in the cluster. Each node is responsible for keeping at least one copy of any partitions assigned to it (that is, at least one replica) available to the cluster.

A replica belongs entirely to a single node; a node can (and usually does) store several replicas.

- **Replica.** This is a copy of a cluster partition. Each node in a node group stores a replica. Also sometimes known as a *partition replica*. The number of replicas is equal to the number of nodes per node group.

The following diagram illustrates a MySQL Cluster with four data nodes, arranged in two node groups of two nodes each; nodes 1 and 2 belong to node group 0, and nodes 3 and 4 belong to node group 1. Note that only data (**ndbd**) nodes are shown here; although a working cluster requires an **ndb_mgm** process for cluster management and at least one SQL node to access the data stored by the cluster, these have been omitted in the figure for clarity.

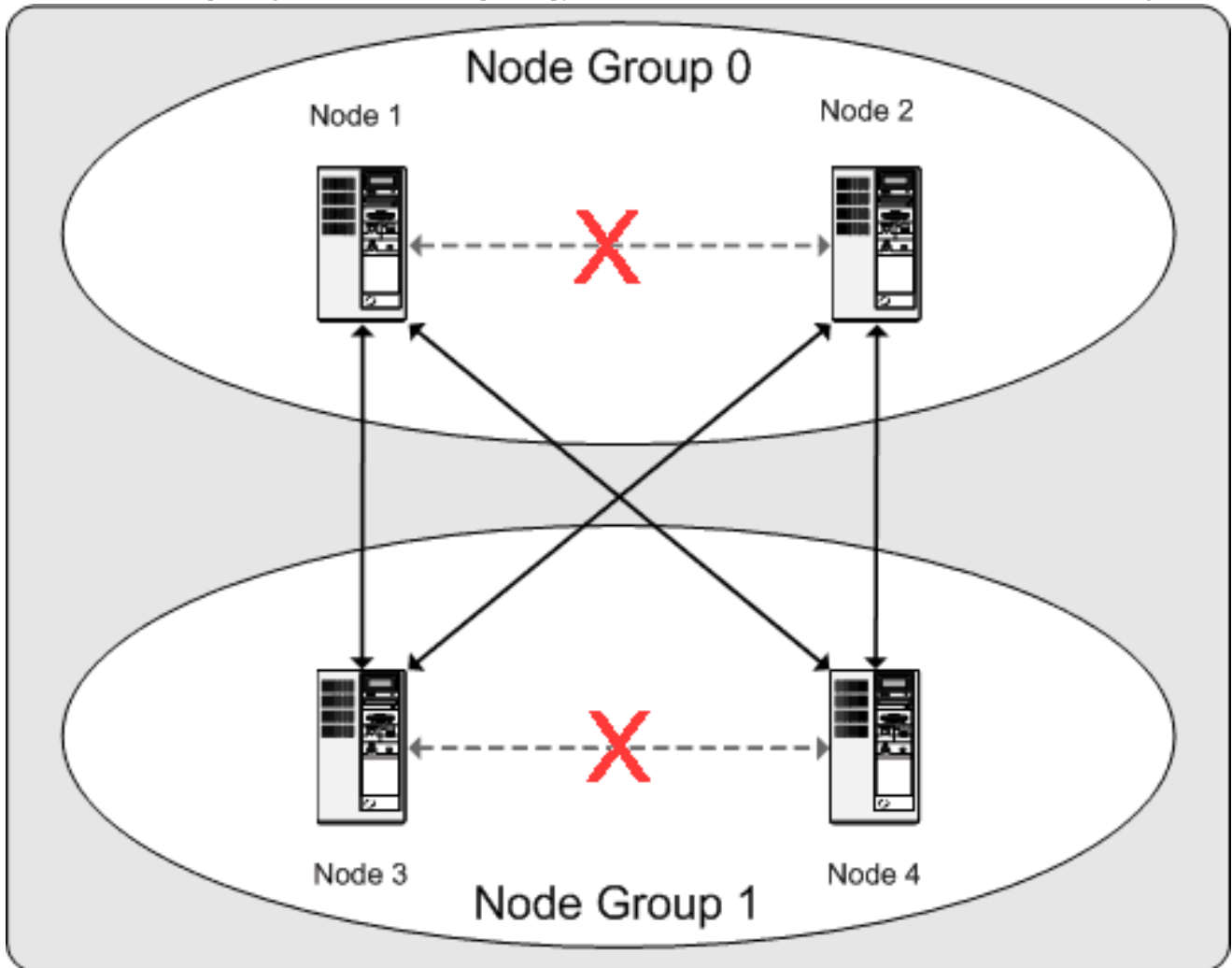


The data stored by the cluster is divided into four partitions, numbered 0, 1, 2, and 3. Each partition is stored — in multiple copies — on the same node group. Partitions are stored on alternate node groups:

- Partition 0 is stored on node group 0; a *primary replica* (primary copy) is stored on node 1, and a *backup replica* (backup copy of the partition) is stored on node 2.
- Partition 1 is stored on the other node group (node group 1); this partition's primary replica is on node 3, and its backup replica is on node 4.
- Partition 2 is stored on node group 0. However, the placing of its two replicas is reversed from that of Partition 0; for Partition 2, the primary replica is stored on node 2, and the backup on node 1.
- Partition 3 is stored on node group 1, and the placement of its two replicas are reversed from those of partition 1. That is, its primary

replica is located on node 4, with the backup on node 3.

What this means regarding the continued operation of a MySQL Cluster is this: so long as each node group participating in the cluster has at least one node operating, the cluster has a complete copy of all data and remains viable. This is illustrated in the next diagram.



In this example, where the cluster consists of two node groups of two nodes each, any combination of at least one node in node group 0 and at least one node in node group 1 is sufficient to keep the cluster “alive” (indicated by arrows in the diagram). However, if *both* nodes from *either* node group fail, the remaining two nodes are not sufficient (shown by the arrows marked out with an X); in either case, the cluster has lost an entire partition and so can no longer provide access to a complete set of all cluster data.

16.2. MySQL Cluster 5.1 Carrier Grade Edition

MySQL Cluster 5.1 Carrier Grade Edition is a branch of MySQL 5.1 using advanced versions of the [NDB](#) storage engine and [NDB](#) API, created as a direct response to customer needs. It is intended for use in the telecommunications industry, and is available in binary and source form to commercial customers.

Two development trees can also be accessed via <http://mysql.bkbits.net/>:

- [mysql-5.1-telco](#)
- [mysql-5.1-telco-6.1](#)

Important

This chapter of the MySQL Manual covers both MySQL 5.1 and MySQL Cluster 5.1 Carrier Grade Edition.

Information which applies to MySQL Cluster 5.1 Carrier Grade Edition releases but not to mainline 5.1 releases is indicated with a warning such as this one:

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

Information which applies to mainline MySQL 5.1 releases but not to MySQL Cluster 5.1 Carrier Grade Edition releases is indicated with a warning such as this one:

MySQL Cluster 5.1 Carrier Grade Edition

The following information does not apply to users of MySQL Cluster 5.1 Carrier Grade Edition. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

(Similar warnings are displayed where appropriate in [The MySQL Cluster API Developers' Guide](#).)

Currently, both the `ndb-6.1.x` and `ndb-6.2.x` series are under active development, with the `ndb-6.1.x` series intended for use by telecommunications customers and `ndb-6.2.x` intended for testing purposes. Differences between the mainline MySQL 5.1 series and MySQL Cluster 5.1 Carrier Grade Edition are highlighted in [Section 16.2.1, “Major Differences Between MySQL 5.1 and MySQL Cluster 5.1 Carrier Grade Edition”](#).

MySQL Cluster 5.1 Carrier Grade Edition versioning. MySQL Cluster 5.1 Carrier Grade Edition — sometimes also referred to as “CGE” — follows a somewhat different release pattern from the mainline MySQL 5.1 Cluster series of releases. Each MySQL Cluster 5.1 Carrier Grade Edition release is identified by a two-part version string which identifies the mainline MySQL version from which the CGE release was branched and the version of the **NDB** storage engine used. For example, the first CGE release was `mysql-5.1.14-ndb-6.1.0` (shown as “MySQL 5.1.14-ndb-6.1.0” in this Manual). The version string tells us that this version:

- Derives from MySQL 5.1.14, and contains all feature enhancement and bugfixes from MySQL 5.1, up to and including MySQL 5.1.14.
- Uses version 6.1.0 of the **NDB** storage engine.

A listing of available 5.1.15-ndb-6.1.6-beta releases for both the `ndb-6.1.x` and the `ndb-6.1.x` series can be found in [Section 16.2.2, “MySQL Cluster 5.1 Carrier Grade Edition Releases”](#).

Additional information about obtaining MySQL Cluster 5.1 Carrier Grade Edition binaries can be found on the MySQL AB web site at <http://www.mysql.com/products/database/clustercge/>, or by contacting [<sales@mysql.com>](mailto:sales@mysql.com).

16.2.1. Major Differences Between MySQL 5.1 and MySQL Cluster 5.1 Carrier Grade Edition

This section lists the most prominent feature differences between mainline releases of MySQL 5.1 and MySQL Cluster 5.1 Carrier Grade Edition releases.

Note

MySQL Cluster 5.1 Carrier Grade Edition also fixes many MySQL Cluster bugs before the fixes appear in MySQL 5.1. The MySQL Cluster 5.1 Carrier Grade Edition changelogs has complete listings of these — see [Section 16.2.2, “MySQL Cluster 5.1 Carrier Grade Edition Releases”](#).

ndb-6.1.x branch.

- Enhanced backup status reporting, aided in part by the introduction of a `BackupReportFrequency` configuration parameter (MySQL 5.1.14-ndb-6.1.0).
- Maximum number of all nodes in a cluster increased to 255 (MySQL 5.1.14-ndb-6.1.1).

- Addition of the `NDB API Dictionary::listEvents()` method (MySQL 5.1.15-ndb-6.1.3).
- Ability to disable arbitration, by setting `ArbitrationRank=0` on all nodes (MySQL 5.1.15-ndb-6.1.3).
- Inclusion of the `ndb_redo_log_reader` utility in the default build (MySQL 5.1.15-ndb-6.1.3).
- New methods of the `Ndb_cluster_connection` class, making it possible to iterate over all existing `Ndb` objects (MySQL 5.1.15-ndb-6.1.4).
- `--ndb-wait-connected` option for `mysqld`, causing `mysqld` wait a specified amount of time to be connected to the cluster before starting to accept client connections (MySQL 5.1.15-ndb-6.1.4).
- Improved data node memory allocation (MySQL 5.1.15-ndb-6.1.4).
- Ability to pipe output of `ndb_restore` to CSV file (MySQL 5.1.15-ndb-6.1.5).
- A new `FragmentLogFileSize` configuration parameter makes it possible to set the size of redo log files (MySQL 5.1.15-ndb-6.1.11).
- `MaxAllocate` configuration parameter makes it possible to set the maximum size of the allocation unit used for table memory (MySQL 5.1.15-ndb-6.1.12).
- New management client `DUMP` commands providing help with tracking transactions, scan operations, and locks (MySQL 5.1.15-ndb-6.1.12).
- Improvements in backups of Disk Data tables resulted in a 10 to 15% increase in backup speed of Disk Data tables (MySQL 5.1.15-ndb-6.1.13).
- Batching of updates on cluster replication slaves, enabled using the `--slave-allow-batching` option for `mysqld` (MySQL 5.1.15-ndb-6.1.17).

ndb-6.2.x branch. The following improvements are available in ndb-6.2.x releases:

- Multiple cluster connections by a single MySQL server using the `--ndb-cluster-connection-pool` startup option for `mysqld` (MySQL 5.1.18-ndb-6.2.2).
- New management client `DUMP` commands providing help with tracking transactions, scan operations, and locks (MySQL 5.1.18-ndb-6.2.2).
- The addition of the `NdbRecord` interface and handler for the `NDB` API (MySQL 5.1.19-ndb-6.2.3).
- Enhanced backup status reporting, aided in part by the introduction of a `BackupReportFrequency` configuration parameter as well as a new management client command `REPORT BackupStatus` (MySQL 5.1.19-ndb-6.2.3).
- In-progress status reporting by `ndb_restore` (MySQL 5.1.19-ndb-6.2.3).
- Batching of updates on cluster replication slaves, enabled using the `--slave-allow-batching` option for `mysqld` (MySQL 5.1.19-ndb-6.2.3).
- Improved memory allocation in the `NDB` kernel (MySQL 5.1.19-ndb-6.2.3).
- Improvements in backups of Disk Data tables resulted in a 10 to 15% increase in backup speed of Disk Data tables (MySQL 5.1.19-ndb-6.2.3).
- `MaxAllocate` configuration parameter makes it possible to set the maximum size of the allocation unit used for table memory (MySQL 5.1.19-ndb-6.2.3).
- The ability to control whether fixed-width or variable-width storage is used for a given column of an `NDB` table by means of the `COLUMN_FORMAT` specifier as part of the column's definition in a `CREATE TABLE` or `ALTER TABLE` statement. In addition, the ability to control whether a given column of an `NDB` table is stored in memory or on disk, using the `STORAGE` specifier as part of the column's definition in a `CREATE TABLE` or `ALTER TABLE` statement. (MySQL 5.1.19-ndb-6.2.5)
- The `--bind-address` cluster management server startup option makes it possible to restrict management client connections to `ndb_mgmd` to a single host and port. (MySQL 5.1.19-ndb-6.2.5)

- Due to a change in the protocol for handling of global checkpoints (GCPs handled in this manner sometimes being referred to as “micro-GCPs”), it is now possible to control how often the GCI number is updated, and how often global checkpoints are written to disk, using the [TimeBetweenEpochs](#) configuration parameter. This improves the reliability and performance of MySQL Cluster Replication. (MySQL 5.1.22-ndb-6.2.5)

ndb-6.3.x branch. The following improvements are available in ndb-6.3.x releases:

- Implementation of conflict resolution for use in multi-master replication (MySQL 5.1.19-ndb-6.3.0).
- The ability to control whether fixed-width or variable-width storage is used for a given column of an **NDB** table by means of the [COLUMN_FORMAT](#) specifier as part of the column's definition in a [CREATE TABLE](#) or [ALTER TABLE](#) statement. In addition, the ability to control whether a given column of an **NDB** table is stored in memory or on disk, using the [STORAGE](#) specifier as part of the column's definition in a [CREATE TABLE](#) or [ALTER TABLE](#) statement. (MySQL 5.1.22-ndb-6.3.2)
- The [--bind-address](#) cluster management server startup option makes it possible to restrict management client connections to [ndb_mgmd](#) to a single host and port. (MySQL 5.1.22-ndb-6.3.2)
- Due to a change in the protocol for handling of global checkpoints (GCPs handled in this manner sometimes being referred to as “micro-GCPs”), it is now possible to control how often the GCI number is updated, and how often global checkpoints are written to disk, using the [TimeBetweenEpochs](#) configuration parameter. This improves the reliability and performance of MySQL Cluster Replication. (MySQL 5.1.22-ndb-6.3.2)

More detailed information can be found in the changelogs for individual MySQL Cluster 5.1 Carrier Grade Edition releases; see [Section 16.2.2, “MySQL Cluster 5.1 Carrier Grade Edition Releases”](#), for a current changelog listing.

16.2.2. MySQL Cluster 5.1 Carrier Grade Edition Releases

Changelogs and source code download locations for MySQL Cluster 5.1 Carrier Grade Edition releases may be found in [Section C.1, “Changes in release 5.1.x \(Development\)”](#); these are grouped together according to the mainline MySQL 5.1 version from which they derive, as shown in the list that follows:

- [Section C.1.16, “Changes in MySQL 5.1.14 Carrier Grade Edition”](#):

Includes [Section C.1.16.1, “Changes in release MySQL 5.1.14-ndb-6.1.0-beta \(20 December 2006\)”](#).

This release includes all feature enhancements and bugfixes made in MySQL 5.1 up to and including the 5.1.14 release.

- [Section C.1.14, “Changes in MySQL 5.1.15 Carrier Grade Edition”](#):

Includes these changelogs:

- [Section C.1.14.19, “Changes in release MySQL 5.1.15-ndb-6.1.1-beta \(01 February 2007\)”](#)
- [Section C.1.14.18, “Changes in release MySQL 5.1.15-ndb-6.1.2-beta \(07 February 2007\)”](#)
- [Section C.1.14.17, “Changes in release MySQL 5.1.15-ndb-6.1.3-beta \(25 February 2007\)”](#)
- [Section C.1.14.16, “Changes in release MySQL 5.1.15-ndb-6.1.4-beta \(09 March 2007 - testing only\)”](#)
- [Section C.1.14.15, “Changes in release MySQL 5.1.15-ndb-6.1.5-beta \(15 March 2007\)”](#)
- [Section C.1.14.14, “Changes in release MySQL 5.1.15-ndb-6.1.6-beta \(Not released\)”](#)
- [Section C.1.14.13, “Changes in release MySQL 5.1.15-ndb-6.1.7-beta \(Released 05 May 2007\)”](#)
- [Section C.1.14.12, “Changes in release MySQL 5.1.15-ndb-6.1.8-beta \(Released 05 May 2007\)”](#)
- [Section C.1.14.11, “Changes in release MySQL 5.1.15-ndb-6.1.9-beta \(Released 24 May 2007\)”](#)
- [Section C.1.14.10, “Changes in release MySQL 5.1.15-ndb-6.1.10-beta \(Released 30 May 2007\)”](#)
- [Section C.1.14.9, “Changes in release MySQL 5.1.15-ndb-6.1.11-beta \(Released 06 June 2007\)”](#)

- [Section C.1.14.8, “Changes in release MySQL 5.1.15-ndb-6.1.12-beta \(Released 13 June 2007\)”](#)
- [Section C.1.14.7, “Changes in release MySQL 5.1.15-ndb-6.1.13-beta \(Released 15 June 2007\)”](#)
- [Section C.1.14.6, “Changes in release MySQL 5.1.15-ndb-6.1.14-beta \(Released 19 June 2007\)”](#)
- [Section C.1.14.5, “Changes in release MySQL 5.1.15-ndb-6.1.15-beta \(Released 20 June 2007\)”](#)
- [Section C.1.14.4, “Changes in release MySQL 5.1.15-ndb-6.1.16-beta \(Released 29 June 2007\)”](#)
- [Section C.1.14.3, “Changes in release MySQL 5.1.15-ndb-6.1.17-beta \(Released 03 July 2007\)”](#)

These releases include all feature enhancements and bugfixes made in MySQL 5.1 up to and including 5.1.15, as well as those enhancements specific to MySQL Cluster 5.1 Carrier Grade Edition made in MySQL-5.1.14-ndb-6.1.0.

- [Section C.1.12, “Changes in MySQL 5.1.16 Carrier Grade Edition”](#):

Includes [Section C.1.12.1, “Changes in release MySQL 5.1.16-ndb-6.2.0-beta \(03 March 2007\)”](#).

This release includes all feature enhancements and bugfixes made in MySQL 5.1 up to and including 5.1.16, as well as those enhancements specific to MySQL Cluster 5.1 Carrier Grade Edition that were made in ndb-6.1.x releases.

- [Section C.1.9, “Changes in MySQL 5.1.18 Carrier Grade Edition”](#):

Includes these changelogs:

- [Section C.1.9.2, “Changes in release MySQL 5.1.18-ndb-6.2.1-beta \(Released 30 April 2007\)”](#) and
- [Section C.1.9.1, “Changes in release MySQL 5.1.18-ndb-6.2.2-beta \(07 May 2007\)”](#).

These releases include all feature enhancements and bugfixes made in MySQL 5.1 up to and including 5.1.18, as well as those enhancements specific to MySQL Cluster 5.1 Carrier Grade Edition that were made in ndb-6.2.1 and ndb-6.2.2.

Note

MySQL 5.1.18-ndb-6.2.1 was withdrawn after release, and is no longer available for download.

- [Section C.1.7, “Changes in MySQL 5.1.19 Carrier Grade Edition”](#):

Includes these changelogs:

- [Section C.1.7.4, “Changes in release MySQL 5.1.19-ndb-6.2.3-beta \(02 July 2007\)”](#)
- [Section C.1.7.3, “Changes in release MySQL 5.1.19-ndb-6.2.4-beta \(04 July 2007\)”](#)
- [Section C.1.7.2, “Changes in release MySQL 5.1.19-ndb-6.3.0-beta \(02 July 2007\)”](#)
- [Section C.1.7.1, “Changes in release MySQL 5.1.19-ndb-6.3.1-beta \(04 July 2007\)”](#)

These releases include all feature enhancements and bugfixes made in MySQL 5.1 up to and including 5.1.19, as well as those enhancements specific to MySQL Cluster 5.1 Carrier Grade Edition that were made in ndb-6.2.x releases beginning with MySQL 5.1.19-ndb-6.2.3, as well as those that were made in ndb-6.2.1 and ndb-6.2.2.

- [Section C.1.3, “Changes in MySQL 5.1.20 Carrier Grade Edition”](#):

Includes these changelogs:

- [Section C.1.3.3, “Changes in release MySQL 5.1.22-ndb-6.2.5-beta \(06 September 2007\)”](#)
- [Section C.1.3.2, “Changes in release MySQL 5.1.22-ndb-6.2.6-beta \(Not yet released\)”](#)
- [Section C.1.3.1, “Changes in release MySQL 5.1.22-ndb-6.3.2-beta \(Not yet released\)”](#)

These releases include all feature enhancements and bugfixes made in MySQL 5.1 up to and including 5.1.22. MySQL 5.1.22-ndb-6.2.5 and later ndb-6.2.x releases based on MySQL 5.1.22 include those enhancements made in MySQL 5.1.18-ndb-6.2.1, MySQL 5.1.18-ndb-6.2.2, MySQL 5.1.19-ndb-6.2.3, and MySQL 5.1.19-ndb-6.2.4; MySQL 5.1.22-based ndb-

6.3.x releases include those enhancements made in MySQL 5.1.19-ndb-6.3.0 and MySQL 5.1.19-ndb-6.3.1.

Each of the MySQL Cluster 5.1 Carrier Grade Edition includes enhancements to the [NDB](#) storage engine that do not appear in the mainline MySQL 5.1 tree. We plan to port these to MySQL 5.1 or later mainline MySQL releases at some point in the future.

Some fixes may be listed more than once in [Section C.1, “Changes in release 5.1.x \(Development\)”](#), due to the fact that they were applied first in MySQL Cluster 5.1 Carrier Grade Edition and then ported to MySQL 5.1, because they were applied in more than one branch of MySQL Cluster 5.1 Carrier Grade Edition, or both.

Since all bugfixes applied in MySQL Cluster 5.1 Carrier Grade Edition relate to MySQL Cluster, changelog entries for CGE releases are not prefixed with “[NDB Cluster:](#)” as MySQL Cluster bugfixes in mainline MySQL 5.1 are.

16.3. Simple Multi-Computer How-To

This section is a “How-To” that describes the basics for how to plan, install, configure, and run a MySQL Cluster. Whereas the examples in [Section 16.4, “MySQL Cluster Configuration”](#) provide more in-depth information on a variety of clustering options and configuration, the result of following the guidelines and procedures outlined here should be a usable MySQL Cluster which meets the *minimum* requirements for availability and safeguarding of data.

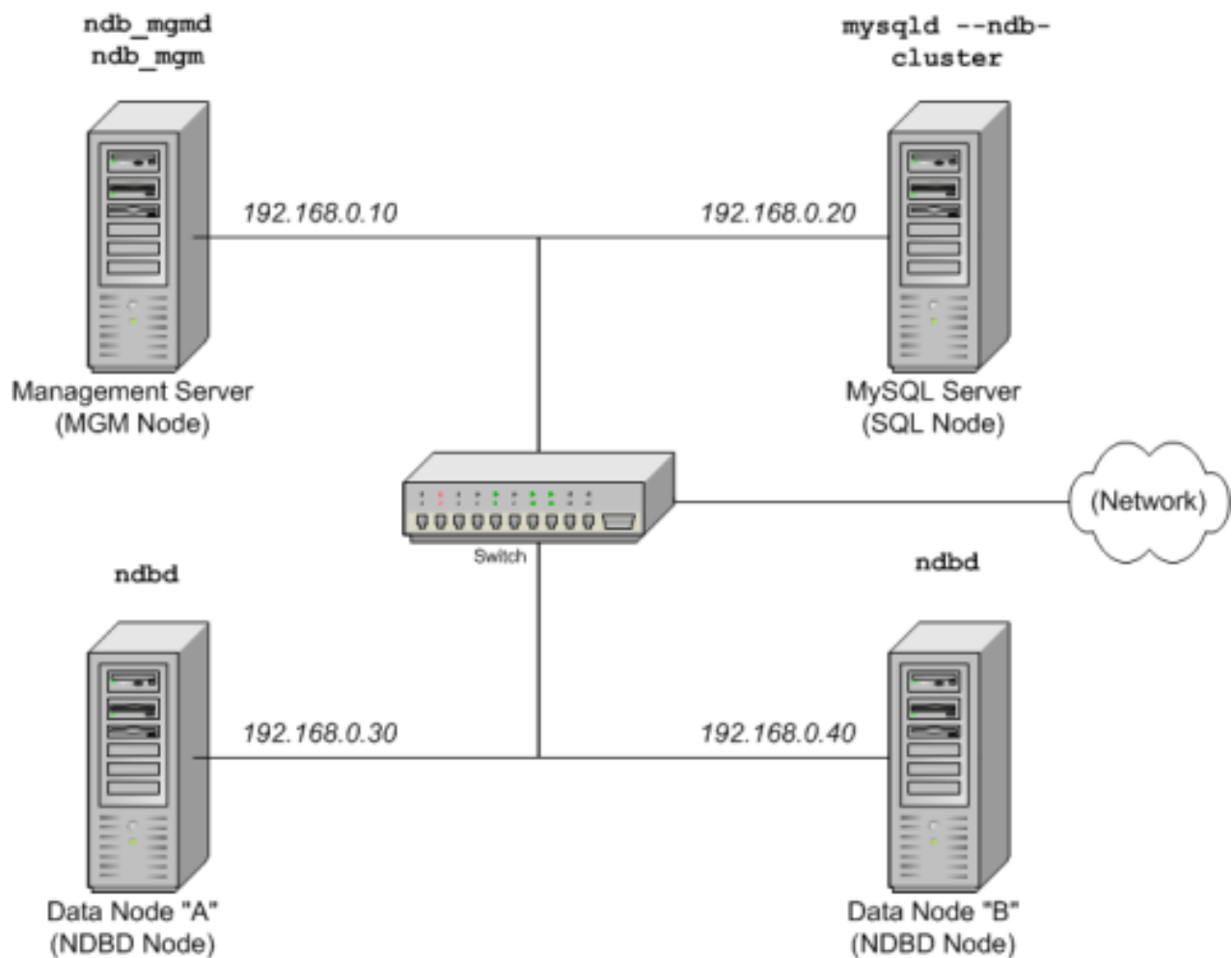
This section covers hardware and software requirements; networking issues; installation of MySQL Cluster; configuration issues; starting, stopping, and restarting the cluster; loading of a sample database; and performing queries.

Basic assumptions. This How-To makes the following assumptions:

1. The cluster setup has four nodes, each on a separate host, and each with a fixed network address on a typical Ethernet as shown here:

Node	IP Address
Management (MGM) node	192.168.0.10
MySQL server (SQL) node	192.168.0.20
Data (NDBD) node "A"	192.168.0.30
Data (NDBD) node "B"	192.168.0.40

This may be made clearer in the following diagram:



In the interest of simplicity (and reliability), this How-To uses only numeric IP addresses. However, if DNS resolution is available on your network, it is possible to use hostnames in lieu of IP addresses in configuring Cluster. Alternatively, you can use the `/etc/hosts` file or your operating system's equivalent for providing a means to do host lookup if such is available.

Note

A common problem when trying to use hostnames for Cluster nodes arises because of the way in which some operating systems (including some Linux distributions) set up the system's own hostname in the `/etc/hosts` during installation. Consider two machines with the hostnames `ndb1` and `ndb2`, both in the `cluster` network domain. Red Hat Linux (including some derivatives such as CentOS and Fedora) places the following entries in these machines' `/etc/hosts` files:

```
# ndb1 /etc/hosts:
127.0.0.1 ndb1.cluster ndb1 localhost.localdomain localhost
```

```
# ndb2 /etc/hosts:
127.0.0.1 ndb2.cluster ndb2 localhost.localdomain localhost
```

SuSE Linux (including OpenSuSE) places these entries in the machines' `/etc/hosts` files:

```
# ndb1 /etc/hosts:
127.0.0.1 localhost
127.0.0.2 ndb1.cluster ndb1
```

```
# ndb2 /etc/hosts:
127.0.0.1 localhost
127.0.0.2 ndb2.cluster ndb2
```

In both instances, `ndb1` routes `ndb1.cluster` to a loopback IP address, but gets a public IP address from DNS for `ndb2.cluster`, while `ndb2` routes `ndb2.cluster` to a loopback address and obtains a public address for `ndb1.cluster`. The result is that each data node connects to the management server, but cannot tell when any other data nodes have connected, and so the data nodes appear to hang while starting.

You should also be aware that you cannot mix `localhost` and other hostnames or IP addresses in `config.ini`. For these reasons, the solution in such cases (other than to use IP addresses for all `config.ini` `HostName` entries) is to remove the fully qualified hostnames from `/etc/hosts` and use these in `config.ini` for all cluster hosts.

- Each host in our scenario is an Intel-based desktop PC running a common, generic Linux distribution installed to disk in a standard configuration, and running no unnecessary services. The core OS with standard TCP/IP networking capabilities should be sufficient. Also for the sake of simplicity, we also assume that the filesystems on all hosts are set up identically. In the event that they are not, you will need to adapt these instructions accordingly.
- Standard 100 Mbps or 1 gigabit Ethernet cards are installed on each machine, along with the proper drivers for the cards, and that all four hosts are connected via a standard-issue Ethernet networking appliance such as a switch. (All machines should use network cards with the same throughput. That is, all four machines in the cluster should have 100 Mbps cards *or* all four machines should have 1 Gbps cards.) MySQL Cluster will work in a 100 Mbps network; however, gigabit Ethernet will provide better performance.

Note that MySQL Cluster is *not* intended for use in a network for which throughput is less than 100 Mbps. For this reason (among others), attempting to run a MySQL Cluster over a public network such as the Internet is not likely to be successful, and is not recommended.

- For our sample data, we will use the `world` database which is available for download from the MySQL AB Web site. As this database takes up a relatively small amount of space, we assume that each machine has 256MB RAM, which should be sufficient for running the operating system, host NDB process, and (for the data nodes) for storing the database.

Although we refer to a Linux operating system in this How-To, the instructions and procedures that we provide here should be easily adaptable to other supported operating systems. We also assume that you already know how to perform a minimal installation and configuration of the operating system with networking capability, or that you are able to obtain assistance in this elsewhere if needed.

We discuss MySQL Cluster hardware, software, and networking requirements in somewhat greater detail in the next section. (See [Section 16.3.1, “Hardware, Software, and Networking”](#).)

16.3.1. Hardware, Software, and Networking

One of the strengths of MySQL Cluster is that it can be run on commodity hardware and has no unusual requirements in this regard, other than for large amounts of RAM, due to the fact that all live data storage is done in memory. (Note that this is not the case with Disk Data tables — see [Section 16.11, “MySQL Cluster Disk Data Tables”](#), for more information about these.) Naturally, multiple and faster CPUs will enhance performance. Memory requirements for other Cluster processes are relatively small.

The software requirements for Cluster are also modest. Host operating systems do not require any unusual modules, services, applications, or configuration to support MySQL Cluster. For supported operating systems, a standard installation should be sufficient. The MySQL software requirements are simple: all that is needed is a production release of MySQL 5.1 to have Cluster support. It is not necessary to compile MySQL yourself merely to be able to use Cluster. In this How-To, we assume that you are using the server binary appropriate to your operating system, available via the MySQL software downloads page at <http://dev.mysql.com/downloads/>.

For inter-node communication, Cluster supports TCP/IP networking in any standard topology, and the minimum expected for each host is a standard 100 Mbps Ethernet card, plus a switch, hub, or router to provide network connectivity for the cluster as a whole. We strongly recommend that a MySQL Cluster be run on its own subnet which is not shared with non-Cluster machines for the following reasons:

- Security.** Communications between Cluster nodes are not encrypted or shielded in any way. The only means of protecting transmissions within a MySQL Cluster is to run your Cluster on a protected network. If you intend to use MySQL Cluster for Web applications, the cluster should definitely reside behind your firewall and not in your network's De-Militarized Zone (DMZ) or elsewhere.
- Efficiency.** Setting up a MySQL Cluster on a private or protected network allows the cluster to make exclusive use of bandwidth between cluster hosts. Using a separate switch for your MySQL Cluster not only helps protect against unauthorized access to Cluster data, it also ensures that Cluster nodes are shielded from interference caused by transmissions between other computers on the network. For enhanced reliability, you can use dual switches and dual cards to remove the network as a single point of failure; many device drivers support failover for such communication links.

It is also possible to use the high-speed Scalable Coherent Interface (SCI) with MySQL Cluster, but this is not a requirement. See [Section 16.12, “Using High-Speed Interconnects with MySQL Cluster”](#), for more about this protocol and its use with MySQL Cluster.

16.3.2. Multi-Computer Installation

Each MySQL Cluster host computer running data or SQL nodes must have installed on it a MySQL server binary. For management nodes, it is not necessary to install the MySQL server binary, but you do have to install the MGM server daemon (`ndb_mgmd`). It is also a good idea to install the management client (`ndb_mgm`) on the management server host. This section covers the steps necessary to install the correct binaries for each type of Cluster node.

MySQL AB provides precompiled binaries that support Cluster, and there is generally no need to compile these yourself. Therefore, the first step in the installation process for each cluster host is to download the file `mysql-5.1.21-beta-pc-linux-gnu-i686.tar.gz` from the [MySQL downloads area](#). We assume that you have placed it in each machine's `/var/tmp` directory. (If you do require a custom binary, see [Section 2.9.3, “Installing from the Development Source Tree”](#).)

MySQL Cluster 5.1 Carrier Grade Edition

The following information does not apply to users of MySQL Cluster 5.1 Carrier Grade Edition. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#).

RPMs are also available for both 32-bit and 64-bit Linux platforms. For a MySQL Cluster, three RPMs are required:

- The **Server** RPM (for example, `MySQL-server-5.1.21-beta-0.glibc23.i386.rpm`), which supplies the core files needed to run a MySQL Server and a MySQL Server binary with clustering support.
- The **NDB Cluster - Storage engine** RPM (for example, `MySQL-ndb-storage-5.1.21-beta-0.glibc23.i386.rpm`), which supplies the MySQL Cluster data node binary (`ndbd`).
- The **NDB Cluster - Storage engine management RPM** (for example, `MySQL-ndb-management-5.1.21-beta-0.glibc23.i386.rpm`), which provides the MySQL Cluster management server binary (`ndb_mgmd`).

In addition, you should also obtain the **NDB Cluster - Storage engine basic tools** RPM (for example, `MySQL-ndb-tools-5.1.21-beta-0.glibc23.i386.rpm`), which supplies several useful applications for working with a MySQL Cluster. The most important of these is the MySQL Cluster management client (`ndb_mgm`). The **NDB Cluster - Storage engine extra tools** RPM (for example, `MySQL-ndb-extra-5.1.21-beta-0.glibc23.i386.rpm`) contains some additional testing and monitoring programs, but is not required to install a MySQL Cluster. (For more information about these additional programs, see [Section 16.9, “Cluster Utility Programs”](#).)

The MySQL version number in the RPM filenames (shown here as `5.1.21-beta`) can vary according to the version which you are actually using. *It is very important that all of the Cluster RPMs to be installed have the same MySQL version number.* The `glibc` version number (if present — shown here as `glibc23`), and architecture designation (shown here as `i386`) should be appropriate to the machine on which the RPM is to be installed.

See [Section 2.4, “Installing MySQL from RPM Packages on Linux”](#), for general information about installing MySQL using RPMs supplied by MySQL AB.

After installing from RPM, you still need to configure the cluster as discussed in [Section 16.3.3, “Multi-Computer Configuration”](#).

The following information applies to all MySQL Cluster users.

Note

After completing the installation, do not yet start any of the binaries. We show you how to do so following the configuration of all nodes.

Data and SQL Node Installation — `.tar.gz` Binary

On each of the machines designated to host data or SQL nodes, perform the following steps as the system `root` user:

1. Check your `/etc/passwd` and `/etc/group` files (or use whatever tools are provided by your operating system for managing users and groups) to see whether there is already a `mysql` group and `mysql` user on the system. Some OS distributions create

these as part of the operating system installation process. If they are not already present, create a new `mysql` user group, and then add a `mysql` user to this group:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

2. Change location to the directory containing the downloaded file, unpack the archive, and create a symlink to the `mysql` directory. Note that the actual file and directory names will vary according to the MySQL version number.

```
shell> cd /var/tmp
shell> tar -C /usr/local -xzf mysql-5.1.21-beta-pc-linux-gnu-i686.tar.gz
shell> ln -s /usr/local/mysql-5.1.21-beta-pc-linux-gnu-i686 /usr/local/mysql
```

3. Change location to the `mysql` directory and run the supplied script for creating the system databases:

```
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
```

4. Set the necessary permissions for the MySQL server and data directories:

```
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .
```

Note that the data directory on each machine hosting a data node is `/usr/local/mysql/data`. We will use this piece of information when we configure the management node. (See [Section 16.3.3, “Multi-Computer Configuration”](#).)

5. Copy the MySQL startup script to the appropriate directory, make it executable, and set it to start when the operating system is booted up:

```
shell> cp support-files/mysql.server /etc/rc.d/init.d/
shell> chmod +x /etc/rc.d/init.d/mysql.server
shell> chkconfig --add mysql.server
```

(The startup scripts directory may vary depending on your operating system and version — for example, in some Linux distributions, it is `/etc/init.d`.)

Here we use Red Hat's `chkconfig` for creating links to the startup scripts; use whatever means is appropriate for this purpose on your operating system and distribution, such as `update-rc.d` on Debian.

Remember that the preceding steps must be performed separately on each machine where a data node or SQL node is to reside.

MySQL Cluster 5.1 Carrier Grade Edition

The following information does not apply to users of MySQL Cluster 5.1 Carrier Grade Edition. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#).

SQL node installation — RPM files. On each machine to be used for hosting a cluster SQL node, install the MySQL RPM by executing the following command as the system root user, replacing the name shown for the RPM as necessary to match the name of the RPM downloaded from the MySQL AB web site:

```
shell> rpm -Uhv MySQL-server-5.1.21-beta-0.glibc23.i386.rpm
```

This installs the MySQL server binary (`mysqld`) in the `/usr/sbin` directory, as well as all needed MySQL Server support files. It also installs the `mysql.server` and `mysqld_safe` startup scripts in `/usr/share/mysql` and `/usr/bin`, respectively. The RPM installer should take care of general configuration issues (such as creating the `mysql` user and group, if needed) automatically.

MySQL Cluster 5.1 Carrier Grade Edition

The following information does not apply to users of MySQL Cluster 5.1 Carrier Grade Edition. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#).

Data Node Installation — RPM Files

On a computer that is to host a cluster data node it is necessary to install only the **NDB Cluster - Storage engine** RPM. To do so, copy this RPM to the data node host, and run the following command as the system root user, replacing the name shown for the RPM as necessary to match that of the RPM downloaded from the MySQL AB web site:

```
shell> rpm -Uvh MySQL-ndb-storage-5.1.21-beta-0.glibc23.i386.rpm
```

The previous command installs the MySQL Cluster data node binary (`ndbd`) in the `/usr/sbin` directory.

The following information applies to all MySQL Cluster users.

Management Node Installation — .tar.gz Binary

Installation for the management (MGM) node does not require installation of the `mysqld` binary. Only the binaries for the MGM server and client are required, which can be found in the downloaded archive. Again, we assume that you have placed this file in `/var/tmp`.

As system `root` (that is, after using `sudo`, `su root`, or your system's equivalent for temporarily assuming the system administrator account's privileges), perform the following steps to install `ndb_mgmd` and `ndb_mgm` on the Cluster management node host:

1. Change location to the `/var/tmp` directory, and extract the `ndb_mgm` and `ndb_mgmd` from the archive into a suitable directory such as `/usr/local/bin`:

```
shell> cd /var/tmp
shell> tar -zxvf mysql-5.1.21-beta-pc-linux-gnu-i686.tar.gz
shell> cd mysql-5.1.21-beta-pc-linux-gnu-i686
shell> cp /bin/ndb_mgm* /usr/local/bin
```

(You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once `ndb_mgm` and `ndb_mgmd` have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

```
shell> cd /usr/local/bin
shell> chmod +x ndb_mgm*
```

MySQL Cluster 5.1 Carrier Grade Edition

The following information does not apply to users of MySQL Cluster 5.1 Carrier Grade Edition. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, "MySQL Cluster 5.1 Carrier Grade Edition".

Management Node Installation — RPM File

To install the MySQL Cluster management server, it is necessary only to use the **NDB Cluster - Storage engine management** RPM. Copy this RPM to the computer intended to host the management node, and then install it by running the following command as the system root user (replace the name shown for the RPM as necessary to match that of the **Storage engine management** RPM downloaded from the MySQL AB web site):

```
shell> rpm -Uvh MySQL-ndb-management-5.1.21-beta-0.glibc23.i386.rpm
```

This installs the management server binary (`ndb_mgmd`) to the `/usr/sbin` directory.

You should also install the **NDB** management client, which is supplied by the **Storage engine basic tools** RPM. Copy this RPM to the same computer as the management node, and then install it by running the following command as the system root user (again, replace the name shown for the RPM as necessary to match that of the **Storage engine basic tools** RPM downloaded from the MySQL AB web site):

```
shell> rpm -Uvh MySQL-ndb-tools-5.1.21-beta-0.glibc23.i386.rpm
```

The **Storage engine basic tools** RPM installs the MySQL Cluster management client (`ndb_mgm`) to the `/usr/bin` directory.

In Section 16.3.3, "Multi-Computer Configuration", we create configuration files for all of the nodes in our example Cluster.

16.3.3. Multi-Computer Configuration

For our four-node, four-host MySQL Cluster, it is necessary to write four configuration files, one per node host.

- Each data node or SQL node requires a `my.cnf` file that provides two pieces of information: a *connectstring* that tells the node where to find the MGM node, and a line telling the MySQL server on this host (the machine hosting the data node) to run in NDB mode.

For more information on connectstrings, see [Section 16.4.4.2, “The Cluster Connectstring”](#).

- The management node needs a `config.ini` file telling it how many replicas to maintain, how much memory to allocate for data and indexes on each data node, where to find the data nodes, where to save data to disk on each data node, and where to find any SQL nodes.

Configuring the Storage and SQL Nodes

The `my.cnf` file needed for the data nodes is fairly simple. The configuration file should be located in the `/etc` directory and can be edited using any text editor. (Create the file if it does not exist.) For example:

```
shell> vi /etc/my.cnf
```

We show `vi` being used here to create the file, but any text editor should work just as well.

For each data node and SQL node in our example setup, `my.cnf` should look like this:

```
# Options for mysqld process:
[MYSQLD]
ndbcluster                # run NDB storage engine
ndb-connectstring=192.168.0.10 # location of management server

# Options for ndbd process:
[MYSQL_CLUSTER]
ndb-connectstring=192.168.0.10 # location of management server
```

After entering the preceding information, save this file and exit the text editor. Do this for the machines hosting data node “A”, data node “B”, and the SQL node.

Important

Once you have started a `mysqld` process with the `ndbcluster` and `ndb-connectstring` parameters in the `[MYSQLD]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. *This is by design.*

Configuring the management node. The first step in configuring the MGM node is to create the directory in which the configuration file can be found and then to create the file itself. For example (running as `root`):

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini
```

For our representative setup, the `config.ini` file should read as follows:

```
# Options affecting ndbd processes on all data nodes:
[NDBD DEFAULT]
NoOfReplicas=2      # Number of replicas
DataMemory=80M      # How much memory to allocate for data storage
IndexMemory=18M     # How much memory to allocate for index storage
                    # For DataMemory and IndexMemory, we have used the
                    # default values. Since the "world" database takes up
                    # only about 500KB, this should be more than enough for
                    # this example cluster setup.

# TCP/IP options:
[TCP DEFAULT]
portnumber=2202     # This the default; however, you can use any
                    # port that is free for all the hosts in the cluster
                    # Note: It is recommended that you do not specify the
                    # portnumber at all and allow the default value to be
                    # used instead

# Management process options:
```

```
[NDB_MGMD]
hostname=192.168.0.10      # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster # Directory for MGM node log files

# Options for data node "A":
[NDBD]
# (one [NDBD] section per data node)
hostname=192.168.0.30      # Hostname or IP address
datadir=/usr/local/mysql/data # Directory for this data node's data files

# Options for data node "B":
[NDBD]
hostname=192.168.0.40      # Hostname or IP address
datadir=/usr/local/mysql/data # Directory for this data node's data files

# SQL node options:
[MYSQLD]
hostname=192.168.0.20      # Hostname or IP address
# (additional mysqld connections can be
# specified for this node for various
# purposes such as running ndb_restore)
```

Note

The `world` database can be downloaded from <http://dev.mysql.com/doc/>, where it can be found listed under “Examples”.

After all the configuration files have been created and these minimal options have been specified, you are ready to proceed with starting the cluster and verifying that all processes are running. We discuss how this is done in [Section 16.3.4, “Initial Startup”](#).

For more detailed information about the available MySQL Cluster configuration parameters and their uses, see [Section 16.4.4, “Configuration File”](#), and [Section 16.4, “MySQL Cluster Configuration”](#). For configuration of MySQL Cluster as relates to making backups, see [Section 16.8.4, “Configuration for Cluster Backup”](#).

Note

The default port for Cluster management nodes is 1186; the default port for data nodes is 2202. However, the cluster can automatically allocate ports for data nodes from those that are already free.

16.3.4. Initial Startup

Starting the cluster is not very difficult after it has been configured. Each cluster node process must be started separately, and on the host where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes:

1. On the management host, issue the following command from the system shell to start the management node process:

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

Note that `ndb_mgmd` must be told where to find its configuration file, using the `-f` or `--config-file` option. (See [Section 16.6.3, “ndb_mgmd — The Management Server Process”](#), for details.)

2. On each of the data node hosts, run this command to start the `ndbd` process:

```
shell> ndbd
```

- 3.

MySQL Cluster 5.1 Carrier Grade Edition

The following information does not apply to users of MySQL Cluster 5.1 Carrier Grade Edition. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#).

If you used RPM files to install MySQL on the cluster host where the SQL node is to reside, you can (and should) use the supplied startup script to start the MySQL server process on the SQL node.

The following information applies to all MySQL Cluster users.

If all has gone well, and the cluster has been set up correctly, the cluster should now be operational. You can test this by invoking the `ndb_mgm` management node client. The output should look like that shown here, although you might see some slight differences in the

output depending upon the exact version of MySQL that you are using:

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=2 @192.168.0.30 (Version: 5.1.21-beta, Nodegroup: 0, Master)
id=3 @192.168.0.40 (Version: 5.1.21-beta, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @192.168.0.10 (Version: 5.1.21-beta)

[mysqld(SQL)] 1 node(s)
id=4 (Version: 5.1.21-beta)
```

Note

The SQL node is referenced here as `[mysqld(API)]`. This is perfectly normal, and reflects the fact that the `mysqld` process is acting as a cluster API node.

You should now be ready to work with databases, tables, and data in MySQL Cluster. See [Section 16.3.5, “Loading Sample Data and Performing Queries”](#), for a brief discussion.

16.3.5. Loading Sample Data and Performing Queries

Working with data in MySQL Cluster is not much different from doing so in MySQL without Cluster. There are two points to keep in mind:

- For a table to be replicated in the cluster, it must use the **NDB Cluster** storage engine. To specify this, use the **ENGINE=NDB** or **ENGINE=NDBCLUSTER** table option. You can add this option when creating the table:

```
CREATE TABLE tbl_name (column_definitions) ENGINE=NDBCLUSTER;
```

Alternatively, for an existing table that uses a different storage engine, use **ALTER TABLE** to change the table to use **NDB Cluster**:

```
ALTER TABLE tbl_name ENGINE=NDBCLUSTER;
```

- Each **NDB** table *must* have a primary key. If no primary key is defined by the user when a table is created, the **NDB Cluster** storage engine automatically generates a hidden one.

Note

This hidden key takes up space just as does any other table index. It is not uncommon to encounter problems due to insufficient memory for accommodating these automatically created indexes.)

If you are importing tables from an existing database using the output of `mysqldump`, you can open the SQL script in a text editor and add the **ENGINE** option to any table creation statements, or replace any existing **ENGINE** (or **TYPE**) options. Suppose that you have the **world** sample database on another MySQL server that does not support MySQL Cluster, and you want to export the **City** table:

```
shell> mysqldump --add-drop-table world City > city_table.sql
```

The resulting `city_table.sql` file will contain this table creation statement (and the **INSERT** statements necessary to import the table data):

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabul',1780000);
```



```
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

You need to make sure that MySQL uses the **NDB** storage engine for this table. There are two ways that this can be accomplished. One of these is to modify the table definition *before* importing it into the Cluster database. Using the **City** table as an example, modify the **ENGINE** option of the definition as follows:

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabul',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

This must be done for the definition of each table that is to be part of the clustered database. The easiest way to accomplish this is to do a search-and-replace on the file that contains the definitions and replace all instances of **TYPE=engine_name** or **ENGINE=en-gine_name** with **ENGINE=NDBCLUSTER**. If you do not want to modify the file, you can use the unmodified file to create the tables, and then use **ALTER TABLE** to change their storage engine. The particulars are given later in this section.

Assuming that you have already created a database named **world** on the SQL node of the cluster, you can then use the **mysql** command-line client to read **city_table.sql**, and create and populate the corresponding table in the usual manner:

```
shell> mysql world < city_table.sql
```

It is very important to keep in mind that the preceding command must be executed on the host where the SQL node is running (in this case, on the machine with the IP address **192.168.0.20**).

To create a copy of the entire **world** database on the SQL node, use **mysqldump** on the non-cluster server to export the database to a file named **world.sql**; for example, in the **/tmp** directory. Then modify the table definitions as just described and import the file into the SQL node of the cluster like this:

```
shell> mysql world < /tmp/world.sql
```

If you save the file to a different location, adjust the preceding instructions accordingly.

It is important to note that **NDB Cluster** in MySQL 5.1 does not support autodiscovery of databases. (See [Section 16.13, “Known Limitations of MySQL Cluster”](#).) This means that, once the **world** database and its tables have been created on one data node, you need to issue the **CREATE DATABASE world** statement (or you may use **CREATE SCHEMA world** instead), followed by **FLUSH TABLES** on each SQL node in the cluster. This causes the node to recognize the database and read its table definitions.

Running **SELECT** queries on the SQL node is no different from running them on any other instance of a MySQL server. To run queries from the command line, you first need to log in to the MySQL Monitor in the usual way (specify the **root** password at the **Enter password:** prompt):

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.1.21-beta

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

We simply use the MySQL server's **root** account and assume that you have followed the standard security precautions for installing a MySQL server, including setting a strong **root** password. For more information, see [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

It is worth taking into account that Cluster nodes do *not* make use of the MySQL privilege system when accessing one another. Setting or changing MySQL user accounts (including the **root** account) effects only applications that access the SQL node, not interaction between nodes.

If you did not modify the **ENGINE** clauses in the table definitions prior to importing the SQL script, you should run the following statements at this point:

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

Selecting a database and running a **SELECT** query against a table in that database is also accomplished in the usual manner, as is exiting the MySQL Monitor:

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
+-----+-----+
| Name      | Population |
+-----+-----+
| Bombay    | 10500000  |
| Seoul     | 9981619   |
| São Paulo | 9968485   |
| Shanghai  | 9696300   |
| Jakarta   | 9604900   |
+-----+-----+
5 rows in set (0.34 sec)

mysql> \q
Bye

shell>
```

Applications that use MySQL can employ standard APIs to access **NDB** tables. It is important to remember that your application must access the SQL node, and not the management or data nodes. This brief example shows how we might execute the **SELECT** statement just shown by using the PHP 5.X **mysqli** extension running on a Web server elsewhere on the network:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
  # connect to SQL node:
  $link = new mysqli('192.168.0.20', 'root', 'root_password', 'world');
  # parameters for mysqli constructor are:
  #   host, user, password, database

  if( mysqli_connect_errno() )
    die("Connect failed: " . mysqli_connect_error());

  $query = "SELECT Name, Population
            FROM City
            ORDER BY Population DESC
            LIMIT 5";

  # if no errors...
  if( $result = $link->query($query) )
  {
    ?>
    <table border="1" width="40%" cellpadding="4" cellspacing="1">
    <tbody>
    <tr>
      <th width="10%">City</th>
      <th>Population</th>
    </tr>
    <?
      # then display the results...
      while($row = $result->fetch_object())
        printf(<tr>\n  <td align="center">%s</td><td>%d</td>\n</tr>\n",
              $row->Name, $row->Population);
    ?>
    </tbody>
    </table>
    <?
      # ...and verify the number of rows that were retrieved
      printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
    }
    else
      # otherwise, tell us what went wrong
      echo mysqli_error();

    # free the result set and the mysqli connection object
```

```
$result->close();  
$link->close();  
?>  
</body>  
</html>
```

We assume that the process running on the Web server can reach the IP address of the SQL node.

In a similar fashion, you can use the MySQL C API, Perl-DBI, Python-mysql, or MySQL AB's own Connectors to perform the tasks of data definition and manipulation just as you would normally with MySQL.

16.3.6. Safe Shutdown and Restart

To shut down the cluster, enter the following command in a shell on the machine hosting the management node:

```
shell> ndb_mgm -e shutdown
```

The `-e` option here is used to pass a command to the `ndb_mgm` client from the shell. See [Section 4.3.1, “Using Options on the Command Line”](#). The command causes the `ndb_mgm`, `ndb_mgmd`, and any `ndbd` processes to terminate gracefully. Any SQL nodes can be terminated using `mysqldadmin shutdown` and other means.

To restart the cluster, run these commands:

- On the management host (`192.168.0.10` in our example setup):

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

- On each of the data node hosts (`192.168.0.30` and `192.168.0.40`):

```
shell> ndbd
```

- On the SQL host (`192.168.0.20`):

```
shell> mysqld_safe &
```

For information on making Cluster backups, see [Section 16.8.2, “Using The Management Client to Create a Backup”](#).

To restore the cluster from backup requires the use of the `ndb_restore` command. This is covered in [Section 16.8.3, “ndb_restore — Restore a Cluster Backup”](#).

More information on configuring MySQL Cluster can be found in [Section 16.4, “MySQL Cluster Configuration”](#).

16.4. MySQL Cluster Configuration

A MySQL server that is part of a MySQL Cluster differs in only one respect from a normal (non-clustered) MySQL server, in that it employs the **NDB Cluster** storage engine. This engine is also referred to simply as **NDB**, and the two forms of the name are synonymous.

To avoid unnecessary allocation of resources, the server is configured by default with the **NDB** storage engine disabled. To enable **NDB**, you must modify the server's `my.cnf` configuration file, or start the server with the `--ndbcluster` option.

The MySQL server is a part of the cluster, so it also must know how to access an MGM node to obtain the cluster configuration data. The default behavior is to look for the MGM node on `localhost`. However, should you need to specify that its location is elsewhere, this can be done in `my.cnf` or on the MySQL server command line. Before the **NDB** storage engine can be used, at least one MGM node must be operational, as well as any desired data nodes.

16.4.1. Building MySQL Cluster from Source Code

NDB, the Cluster storage engine, is available in binary distributions for Linux, Mac OS X, and Solaris. We are working to make Cluster run on all operating systems supported by MySQL, including Windows.

If you choose to build from a source tarball or the MySQL 5.1 BitKeeper tree, be sure to use the `--with-ndbcluster` option when

running `configure`. You can also use the `BUILD/compile-pentium-max` build script. Note that this script includes OpenSSL, so you must either have or obtain OpenSSL to build successfully, or else modify `compile-pentium-max` to exclude this requirement. Of course, you can also just follow the standard instructions for compiling your own binaries, and then perform the usual tests and installation procedure. See [Section 2.9.3, “Installing from the Development Source Tree”](#).

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#).

Warning

`BUILD/compile-pentium-max` by default builds the MySQL Server with support for the `InnoDB` storage engine. *You must not deploy or make use of MySQL MySQL Cluster 5.1 Carrier Grade Edition employing the `InnoDB` storage engine without prior written permission to do so from MySQL AB.*

If you wish to use `InnoDB` technology in conjunction with your use of MySQL MySQL Cluster 5.1 Carrier Grade Edition and do not already have authorization to do so as part of your MySQL Server license, contact your MySQL AB salesperson or other authorized MySQL AB representative for assistance in upgrading your MySQL Server license.

The following information applies to all MySQL Cluster users.

`BUILD/compile-pentium-max` also includes OpenSSL, so you must either have or obtain OpenSSL to build successfully, or else modify `compile-pentium-max` to exclude this requirement. Of course, you can also just follow the standard instructions for compiling your own binaries, and then perform the usual tests and installation procedure. See [Section 2.9.3, “Installing from the Development Source Tree”](#).

You should also note that `compile-pentium-max` installs MySQL to the directory `/usr/local/mysql`, placing all MySQL Cluster executables, scripts, databases, and support files in subdirectories under this directory. If this is not what you desire, be sure to modify the script accordingly.

16.4.2. Installing the Cluster Software

In the next few sections, we assume that you are already familiar with installing MySQL, and here we cover only the differences between configuring MySQL Cluster and configuring MySQL without clustering. (See [Chapter 2, *Installing and Upgrading MySQL*](#), if you require more information about the latter.)

You will find Cluster configuration easiest if you have already have all management and data nodes running first; this is likely to be the most time-consuming part of the configuration. Editing the `my.cnf` file is fairly straightforward, and this section will cover only any differences from configuring MySQL without clustering.

16.4.3. Quick Test Setup of MySQL Cluster

To familiarize you with the basics, we will describe the simplest possible configuration for a functional MySQL Cluster. After this, you should be able to design your desired setup from the information provided in the other relevant sections of this chapter.

First, you need to create a configuration directory such as `/var/lib/mysql-cluster`, by executing the following command as the system `root` user:

```
shell> mkdir /var/lib/mysql-cluster
```

In this directory, create a file named `config.ini` that contains the following information. Substitute appropriate values for `Host-Name` and `DataDir` as necessary for your system.

```
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not required
# to do so.
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: DB, API, and MGM are aliases for NDBD, MYSQLD, and NDB_MGMD
# respectively. DB and API are deprecated and should not be used in new
# installations.
[NDBD DEFAULT]
NoOfReplicas= 1
```

```
[MYSQLD DEFAULT]
[NDB_MGMD DEFAULT]
[TCP DEFAULT]

[NDB_MGMD]
HostName= myhost.example.com

[NDBD]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster

[MYSQLD]
[MYSQLD]
[MYSQLD]
```

You can now start the `ndb_mgmd` management server. By default, it attempts to read the `config.ini` file in its current working directory, so change location into the directory where the file is located and then invoke `ndb_mgmd`:

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

Then start a single data node by running `ndbd`:

```
shell> ndbd
```

By default, `ndbd` looks for the management server at `localhost` on port 1186.

Note

If you have installed MySQL from a binary tarball, you will need to specify the path of the `ndb_mgmd` and `ndbd` servers explicitly. (Normally, these will be found in `/usr/local/mysql/bin`.)

Finally, change location to the MySQL data directory (usually `/var/lib/mysql` or `/usr/local/mysql/data`), and make sure that the `my.cnf` file contains the option necessary to enable the NDB storage engine:

```
[mysqld]
ndbcluster
```

You can now start the MySQL server as usual:

```
shell> mysqld_safe --user=mysql &
```

Wait a moment to make sure the MySQL server is running properly. If you see the notice `mysql ended`, check the server's `.err` file to find out what went wrong.

If all has gone well so far, you now can start using the cluster. Connect to the server and verify that the `NDBCLUSTER` storage engine is enabled:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.1.21-beta

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW ENGINES\G
...
***** 12. row *****
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
***** 13. row *****
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
...
```

The row numbers shown in the preceding example output may be different from those shown on your system, depending upon how your server is configured.

Try to create an `NDBCLUSTER` table:

```
shell> mysql
```

```
mysql> USE test;
Database changed

mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW CREATE TABLE ctest \G
***** 1. row *****
      Table: ctest
Create Table: CREATE TABLE `ctest` (
  `i` int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

To check that your nodes were set up properly, start the management client:

```
shell> ndb_mgm
```

Use the **SHOW** command from within the management client to obtain a report on the cluster's status:

```
ndb_mgm> SHOW
Cluster Configuration
-----
[ndbd(NDB)] 1 node(s)
id=2 @127.0.0.1 (Version: 3.5.3, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @127.0.0.1 (Version: 3.5.3)

[mysqld(API)] 3 node(s)
id=3 @127.0.0.1 (Version: 3.5.3)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)
```

At this point, you have successfully set up a working MySQL Cluster. You can now store data in the cluster by using any table created with **ENGINE=NDBCLUSTER** or its alias **ENGINE=NDB**.

16.4.4. Configuration File

Configuring MySQL Cluster requires working with two files:

- **my.cnf**: Specifies options for all MySQL Cluster executables. This file, with which you should be familiar with from previous work with MySQL, must be accessible by each executable running in the cluster.
- **config.ini**: This file is read only by the MySQL Cluster management server, which then distributes the information contained therein to all processes participating in the cluster. **config.ini** contains a description of each node involved in the cluster. This includes configuration parameters for data nodes and configuration parameters for connections between all nodes in the cluster. For a quick reference to the sections that can appear in this file, and what sorts of configuration parameters may be placed in each section, see [Sections of the config.ini File](#).

We are continuously making improvements in Cluster configuration and attempting to simplify this process. Although we strive to maintain backward compatibility, there may be times when introduce an incompatible change. In such cases we will try to let Cluster users know in advance if a change is not backward compatible. If you find such a change and we have not documented it, please report it in the MySQL bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#).

16.4.4.1. Basic Example Configuration

To support MySQL Cluster, you will need to update **my.cnf** as shown in the following example. Note that the options shown here should not be confused with those that are used in **config.ini** files. You may also specify these parameters on the command line when invoking the executables.

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (valid in MySQL 5.1)

# enable ndbcluster storage engine, and provide connectstring for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com
```

```
# provide connectstring for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connectstring for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com

# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
```

(For more information on connectstrings, see [Section 16.4.4.2, “The Cluster Connectstring”](#).)

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (will work on all versions)

# enable ndbcluster storage engine, and provide connectstring for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Important

Once you have started a `mysqld` process with the `ndbcluster` and `ndb-connectstring` parameters in the `[MYSQLD]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. *This is by design.*

You may also use a separate `[mysql_cluster]` section in the cluster `my.cnf` file for settings to be read and used by all executables:

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

For additional `NDB` variables that can be set in the `my.cnf` file, see [Section 5.2.3, “System Variables”](#).

The configuration file is named `config.ini` by default. It is read by `ndb_mgmd` at startup and can be placed anywhere. Its location and name are specified by using `--config-file=path_name` on the `ndb_mgmd` command line. If the configuration file is not specified, `ndb_mgmd` by default tries to read a file named `config.ini` located in the current working directory.

Currently, the configuration file is in INI format, which consists of sections preceded by section headings (surrounded by square brackets), followed by the appropriate parameter names and values. One deviation from the standard INI format is that the parameter name and value can be separated by a colon (':') as well as the equals sign ('='). Another deviation is that sections are not uniquely identified by section name. Instead, unique sections (such as two different nodes of the same type) are identified by a unique ID specified as a parameter within the section.

Default values are defined for most parameters, and can also be specified in `config.ini`. To create a default value section, simply add the word `DEFAULT` to the section name. For example, an `[NDBD]` section contains parameters that apply to a particular data node, whereas an `[NDBD DEFAULT]` section contains parameters that apply to all data nodes. Suppose that all data nodes should use the same data memory size. To configure them all, create an `[NDBD DEFAULT]` section that contains a `DataMemory` line to specify the data memory size.

At a minimum, the configuration file must define the computers and nodes involved in the cluster and on which computers these nodes are located. An example of a simple configuration file for a cluster consisting of one management server, two data nodes and two MySQL servers is shown here:

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the
# management server)
# The first MySQL Server can be started from any host. The second
# can be started only on the host mysqld_5.mysql.com

[NDBD DEFAULT]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster

[NDB_MGMD]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster

[NDBD]
```

```

HostName= ndbd_2.mysql.com

[NDBD]
HostName= ndbd_3.mysql.com

[MYSQLD]
[MYSQLD]
HostName= mysqld_5.mysql.com

```

Each node has its own section in the `config.ini` file. For example, this cluster has two data nodes, so the preceding configuration file contains two `[NDBD]` sections defining these nodes.

Note

Do not place comments on the same line as a section heading in the `config.ini` file; this causes the management server not to start because it cannot parse the configuration file in such cases.

Sections of the `config.ini` File

There are six different sections that you can use in the `config.ini` configuration file, as described in the following list:

- `[COMPUTER]`: Defines cluster hosts. This is not required to configure a viable MySQL Cluster, but it may be used as a convenience when setting up a large cluster. See [Section 16.4.4.3, “Defining Cluster Computers”](#), for more information.
- `[NDBD]`: Defines a cluster data node (`ndbd` process). See [Section 16.4.4.5, “Defining Data Nodes”](#), for details.
- `[MYSQLD]`: Defines the cluster's MySQL server nodes (also called SQL or API nodes). For a discussion of SQL node configuration, see [Section 16.4.4.6, “Defining SQL and Other API Nodes”](#).
- `[MGM]` or `[NDB_MGMD]`: Defines a cluster management server (MGM) node. For information concerning the configuration of MGM nodes, see [Section 16.4.4.4, “Defining the Management Server”](#).
- `[TCP]`: Defines a TCP/IP connection between cluster nodes, with TCP/IP being the default connection protocol. Normally, `[TCP]` or `[TCP_DEFAULT]` sections are not required to set up a MySQL Cluster, as the cluster handles this automatically; however, it may be necessary in some situations to override the defaults provided by the cluster. See [Section 16.4.4.7, “Cluster TCP/IP Connections”](#), for information about available TCP/IP configuration parameters and how to use them. (You may also find [Section 16.4.4.8, “TCP/IP Connections Using Direct Connections”](#) to be of interest in some cases.)
- `[SHM]`: Defines shared-memory connections between nodes. In MySQL 5.1, it is enabled by default, but should still be considered experimental. For a discussion of SHM interconnects, see [Section 16.4.4.9, “Shared-Memory Connections”](#).
- `[SCI]`: Defines *Scalable Coherent Interface* connections between cluster data nodes. Such connections require software which, while freely available, is not part of the MySQL Cluster distribution, as well as specialised hardware. See [Section 16.4.4.10, “SCI Transport Connections”](#) for detailed information about SCI interconnects.

You can define `DEFAULT` values for each section. All Cluster parameter names are case-insensitive, which differs from parameters specified in `my.cnf` or `my.ini` files.

16.4.4.2. The Cluster Connectstring

With the exception of the MySQL Cluster management server (`ndb_mgmd`), each node that is part of a MySQL Cluster requires a *connectstring* that points to the management server's location. This connectstring is used in establishing a connection to the management server as well as in performing other tasks depending on the node's role in the cluster. The syntax for a connectstring is as follows:

```

<connectstring> :=
    [<nodeid-specification>,<host-specification>[,<host-specification>]
<nodeid-specification> := node_id
<host-specification> := host_name[:port_num]

```

`node_id` is an integer larger than 1 which identifies a node in `config.ini`. `host_name` is a string representing a valid Internet host name or IP address. `port_num` is an integer referring to a TCP/IP port number.

```

example 1 (long):    "nodeid=2,myhost1:1100,myhost2:1100,192.168.0.3:1200"
example 2 (short):   "myhost1"

```


All nodes will use `localhost:1186` as the default connectstring value if none is provided. If `port_num` is omitted from the connectstring, the default port is 1186. This port should always be available on the network because it has been assigned by IANA for this purpose (see <http://www.iana.org/assignments/port-numbers> for details).

By listing multiple `<host-specification>` values, it is possible to designate several redundant management servers. A cluster node will attempt to contact successive management servers on each host in the order specified, until a successful connection has been established.

There are a number of different ways to specify the connectstring:

- Each executable has its own command-line option which enables specifying the management server at startup. (See the documentation for the respective executable.)
- It is also possible to set the connectstring for all nodes in the cluster at once by placing it in a `[mysql_cluster]` section in the management server's `my.cnf` file.
- For backward compatibility, two other options are available, using the same syntax:
 1. Set the `NDB_CONNECTSTRING` environment variable to contain the connectstring.
 2. Write the connectstring for each executable into a text file named `Ndb.cfg` and place this file in the executable's startup directory.

However, these are now deprecated and should not be used for new installations.

The recommended method for specifying the connectstring is to set it on the command line or in the `my.cnf` file for each executable.

The maximum length of a connectstring is 1024 characters.

16.4.4.3. Defining Cluster Computers

The `[COMPUTER]` section has no real significance other than serving as a way to avoid the need of defining host names for each node in the system. All parameters mentioned here are required.

- `Id`

This is an integer value, used to refer to the host computer elsewhere in the configuration file. This is not the same as the node ID.

- `HostName`

This is the computer's hostname or IP address.

16.4.4.4. Defining the Management Server

The `[NDB_MGMD]` section is used to configure the behavior of the management server. `[MGM]` can be used as an alias; the two section names are equivalent. All parameters in the following list are optional and assume their default values if omitted.

Note

If neither the `ExecuteOnComputer` nor the `HostName` parameter is present, the default value `localhost` will be assumed for both.

- `Id`

Each node in the cluster has a unique identity, which is represented by an integer value in the range 1 to 63 inclusive. This ID is used by all internal cluster messages for addressing the node.

- `ExecuteOnComputer`

This refers to the `Id` set for one of the computers defined in a `[COMPUTER]` section of the `config.ini` file.

- **PortNumber**

This is the port number on which the management server listens for configuration requests and management commands.

- **HostName**

Specifying this parameter defines the hostname of the computer on which the management node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- **LogDestination**

This parameter specifies where to send cluster logging information. There are three options in this regard — `CONSOLE`, `SYSLOG`, and `FILE` — with `FILE` being the default:

- `CONSOLE` outputs the log to `stdout`:

```
CONSOLE
```

- `SYSLOG` sends the log to a `syslog` facility, possible values being one of `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `news`, `syslog`, `user`, `uucp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, or `local7`.

Note

Not every facility is necessarily supported by every operating system.

```
SYSLOG:facility=syslog
```

- `FILE` pipes the cluster log output to a regular file on the same machine. The following values can be specified:

- `filename`: The name of the log file.
- `maxsize`: The maximum size (in bytes) to which the file can grow before logging rolls over to a new file. When this occurs, the old log file is renamed by appending `.N` to the filename, where `N` is the next number not yet used with this name.
- `maxfiles`: The maximum number of log files.

```
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
```

The default value for the `FILE` parameter is

`FILE:filename=ndb_node_id_cluster.log,maxsize=1000000,maxfiles=6`, where `node_id` is the ID of the node.

It is possible to specify multiple log destinations separated by semicolons as shown here:

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

- **ArbitrationRank**

This parameter is used to define which nodes can act as arbitrators. Only management nodes and SQL nodes can be arbitrators. `ArbitrationRank` can take one of the following values:

- `0`: The node will never be used as an arbitrator.
- `1`: The node has high priority; that is, it will be preferred as an arbitrator over low-priority nodes.
- `2`: Indicates a low-priority node which be used as an arbitrator only if a node with a higher priority is not available for that purpose.

Normally, the management server should be configured as an arbitrator by setting its `ArbitrationRank` to 1 (the default value) and that of all SQL nodes to 0.

Beginning with MySQL 5.1.16 (*MySQL Cluster 5.1 Carrier Grade Edition*: MySQL 5.1.15-ndb-6.1.3), it is possible to disable arbitration completely by setting `ArbitrationRank` to 0 on all management and SQL nodes.

- **ArbitrationDelay**

An integer value which causes the management server's responses to arbitration requests to be delayed by that number of milliseconds. By default, this value is 0; it is normally not necessary to change it.

- **DataDir**

This specifies the directory where output files from the management server will be placed. These files include cluster log files, process output files, and the daemon's process ID (PID) file. (For log files, this location can be overridden by setting the `FILE` parameter for `LogDestination` as discussed previously in this section.)

The default value for this parameter is the directory in which `ndb_mgmd` is located.

16.4.4.5. Defining Data Nodes

The `[NDBD]` and `[NDBD DEFAULT]` sections are used to configure the behavior of the cluster's data nodes. There are many parameters which control buffer sizes, pool sizes, timeouts, and so forth. The only mandatory parameters are:

- Either `ExecuteOnComputer` or `HostName`, which must be defined in the local `[NDBD]` section.
- The parameter `NoOfReplicas`, which must be defined in the `[NDBD DEFAULT]` section, as it is common to all Cluster data nodes.

Most data node parameters are set in the `[NDBD DEFAULT]` section. Only those parameters explicitly stated as being able to set local values are allowed to be changed in the `[NDBD]` section. Where present, `HostName`, `Id` and `ExecuteOnComputer` *must* be defined in the local `[NDBD]` section, and not in any other section of `config.ini`. In other words, settings for these parameters are specific to one data node.

For those parameters affecting memory usage or buffer sizes, it is possible to use `K`, `M`, or `G` as a suffix to indicate units of 1024, 1024×1024, or 1024×1024×1024. (For example, `100K` means $100 \times 1024 = 102400$.) Parameter names and values are currently case-sensitive.

Identifying data nodes. The `Id` value (that is, the data node identifier) can be allocated on the command line when the node is started or in the configuration file.

- **Id**

This is the node ID used as the address of the node for all cluster internal messages. This is an integer in the range 1 to 63 inclusive. Each node in the cluster must have a unique identity.

- **ExecuteOnComputer**

This refers to the `Id` set for one of the computers defined in a `[COMPUTER]` section.

- **HostName**

Specifying this parameter defines the hostname of the computer on which the data node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

ServerPort (OBSOLETE)

Each node in the cluster uses a port to connect to other nodes. This port is used also for non-TCP transporters in the connection setup phase. The default port is allocated dynamically in such a way as to ensure that no two nodes on the same computer receive the same port number, so it should not normally be necessary to specify a value for this parameter.

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

TcpBind_INADDR_ANY

Setting this parameter to **TRUE** or **1** binds **IP_ADDR_ANY** so that connections can be made from anywhere (for autogenerated connections). The default is **FALSE** (0).

This parameter was added in MySQL 5.1.16-ndb-6.2.0.

The following information applies to all MySQL Cluster users.

NoOfReplicas

This global parameter can be set only in the **[NDBD DEFAULT]** section, and defines the number of replicas for each table stored in the cluster. This parameter also specifies the size of node groups. A node group is a set of nodes all storing the same information.

Node groups are formed implicitly. The first node group is formed by the set of data nodes with the lowest node IDs, the next node group by the set of the next lowest node identities, and so on. By way of example, assume that we have 4 data nodes and that **NoOfReplicas** is set to 2. The four data nodes have node IDs 2, 3, 4 and 5. Then the first node group is formed from nodes 2 and 3, and the second node group by nodes 4 and 5. It is important to configure the cluster in such a manner that nodes in the same node groups are not placed on the same computer because a single hardware failure would cause the entire cluster to crash.

If no node IDs are provided, the order of the data nodes will be the determining factor for the node group. Whether or not explicit assignments are made, they can be viewed in the output of the management client's **SHOW** statement.

There is no default value for **NoOfReplicas**; the maximum possible value is 4.

Important

The value for this parameter must divide evenly into the number of data nodes in the cluster. For example, if there are two data nodes, then **NoOfReplicas** must be equal to either 1 or 2, since 2/3 and 2/4 both yield fractional values; if there are four data nodes, then **NoOfReplicas** must be equal to 1, 2, or 4.

DataDir

This parameter specifies the directory where trace files, log files, pid files and error logs are placed.

FileSystemPath

This parameter specifies the directory where all files created for metadata, REDO logs, UNDO logs (for Disk Data tables) and data files are placed. The default is the directory specified by **DataDir**.

Note

This directory must exist before the **ndbd** process is initiated.

The recommended directory hierarchy for MySQL Cluster includes **/var/lib/mysql-cluster**, under which a directory for the node's filesystem is created. The name of this subdirectory contains the node ID. For example, if the node ID is 2, this subdirectory is named **ndb_2_fs**.

BackupDataDir

This parameter specifies the directory in which backups are placed. If omitted, the default backup location is the directory named **BACKUP** under the location specified by the **FilePath** parameter. (See above.)

Data Memory, Index Memory, and String Memory

DataMemory and **IndexMemory** are [NDBD] parameters specifying the size of memory segments used to store the actual records and their indexes. In setting values for these, it is important to understand how **DataMemory** and **IndexMemory** are used, as they usually need to be updated to reflect actual usage by the cluster:

- **DataMemory**

This parameter defines the amount of space (in bytes) available for storing database records. The entire amount specified by this value is allocated in memory, so it is extremely important that the machine has sufficient physical memory to accommodate it.

The memory allocated by **DataMemory** is used to store both the actual records and indexes. There is a 16-byte overhead on each record; an additional amount for each record is incurred because it is stored in a 32KB page with 128 byte page overhead (see below). There is also a small amount wasted per page due to the fact that each record is stored in only one page.

For variable-size table attributes in MySQL 5.1, the data is stored on separate datapages, allocated from **DataMemory**. Variable-length records use a fixed-size part with an extra overhead of 4 bytes to reference the variable-size part. The variable-size part has 2 bytes overhead plus 2 bytes per attribute.

The maximum record size is currently 8052 bytes.

The memory space defined by **DataMemory** is also used to store ordered indexes, which use about 10 bytes per record. Each table row is represented in the ordered index. A common error among users is to assume that all indexes are stored in the memory allocated by **IndexMemory**, but this is not the case: Only primary key and unique hash indexes use this memory; ordered indexes use the memory allocated by **DataMemory**. However, creating a primary key or unique hash index also creates an ordered index on the same keys, unless you specify **USING HASH** in the index creation statement. This can be verified by running **ndb_desc -d db_name table_name** in the management client.

The memory space allocated by **DataMemory** consists of 32KB pages, which are allocated to table fragments. Each table is normally partitioned into the same number of fragments as there are data nodes in the cluster. Thus, for each node, there are the same number of fragments as are set in **NoOfReplicas**.

Once a page has been allocated, it is currently not possible to return it to the pool of free pages, except by deleting the table. (This also means that **DataMemory** pages, once allocated to a given table, cannot be used by other tables.) Performing a node recovery also compresses the partition because all records are inserted into empty partitions from other live nodes.

The **DataMemory** memory space also contains UNDO information: For each update, a copy of the unaltered record is allocated in the **DataMemory**. There is also a reference to each copy in the ordered table indexes. Unique hash indexes are updated only when the unique index columns are updated, in which case a new entry in the index table is inserted and the old entry is deleted upon commit. For this reason, it is also necessary to allocate enough memory to handle the largest transactions performed by applications using the cluster. In any case, performing a few large transactions holds no advantage over using many smaller ones, for the following reasons:

- Large transactions are not any faster than smaller ones
- Large transactions increase the number of operations that are lost and must be repeated in event of transaction failure
- Large transactions use more memory

The default value for **DataMemory** is 80MB; the minimum is 1MB. There is no maximum size, but in reality the maximum size has to be adapted so that the process does not start swapping when the limit is reached. This limit is determined by the amount of physical RAM available on the machine and by the amount of memory that the operating system may commit to any one process. 32-bit operating systems are generally limited to 2–4GB per process; 64-bit operating systems can use more. For large databases, it may be preferable to use a 64-bit operating system for this reason.

- **IndexMemory**

This parameter controls the amount of storage used for hash indexes in MySQL Cluster. Hash indexes are always used for primary

key indexes, unique indexes, and unique constraints. Note that when defining a primary key and a unique index, two indexes will be created, one of which is a hash index used for all tuple accesses as well as lock handling. It is also used to enforce unique constraints.

The size of the hash index is 25 bytes per record, plus the size of the primary key. For primary keys larger than 32 bytes another 8 bytes is added.

The default value for `IndexMemory` is 18MB. The minimum is 1MB.

- ### `StringMemory`

This parameter determines how much memory is allocated for strings such as table names, and is specified in an `[NDBD]` or `[NDBD DEFAULT]` section of the `config.ini` file. A value between 0 and 100 inclusive is interpreted as a percent of the maximum default value, which is calculated based on a number of factors including the number of tables, maximum table name size, maximum size of `.FRM` files, `MaxNoOfTriggers`, maximum column name size, and maximum default column value. In general it is safe to assume that the maximum default value is approximately 5 MB for a MySQL Cluster having 1000 tables.

A value greater than 100 is interpreted as a number of bytes.

The default value is 5 — that is, 5 percent of the default maximum, or roughly 5 KB. (Note that this is a change from previous versions of MySQL Cluster.)

Under most circumstances, the default value should be sufficient, but when you have a great many Cluster tables (1000 or more), it is possible to get Error 773 `OUT OF STRING MEMORY, PLEASE MODIFY STRINGMEMORY CONFIG PARAMETER: PERMANENT ERROR: SCHEMA ERROR`, in which case you should increase this value. 25 (25 percent) is not excessive, and should prevent this error from recurring in all but the most extreme conditions.

The following example illustrates how memory is used for a table. Consider this table definition:

```
CREATE TABLE example (  
  a INT NOT NULL,  
  b INT NOT NULL,  
  c INT NOT NULL,  
  PRIMARY KEY(a),  
  UNIQUE(b)  
) ENGINE=NDBCLUSTER;
```

For each record, there are 12 bytes of data plus 12 bytes overhead. Having no nullable columns saves 4 bytes of overhead. In addition, we have two ordered indexes on columns `a` and `b` consuming roughly 10 bytes each per record. There is a primary key hash index on the base table using roughly 29 bytes per record. The unique constraint is implemented by a separate table with `b` as primary key and `a` as a column. This other table consumes an additional 29 bytes of index memory per record in the `example` table as well 8 bytes of record data plus 12 bytes of overhead.

Thus, for one million records, we need 58MB for index memory to handle the hash indexes for the primary key and the unique constraint. We also need 64MB for the records of the base table and the unique index table, plus the two ordered index tables.

You can see that hash indexes takes up a fair amount of memory space; however, they provide very fast access to the data in return. They are also used in MySQL Cluster to handle uniqueness constraints.

Currently, the only partitioning algorithm is hashing and ordered indexes are local to each node. Thus, ordered indexes cannot be used to handle uniqueness constraints in the general case.

An important point for both `IndexMemory` and `DataMemory` is that the total database size is the sum of all data memory and all index memory for each node group. Each node group is used to store replicated information, so if there are four nodes with two replicas, there will be two node groups. Thus, the total data memory available is $2 \times \text{DataMemory}$ for each data node.

It is highly recommended that `DataMemory` and `IndexMemory` be set to the same values for all nodes. Data distribution is even over all nodes in the cluster, so the maximum amount of space available for any node can be no greater than that of the smallest node in the cluster.

`DataMemory` and `IndexMemory` can be changed, but decreasing either of these can be risky; doing so can easily lead to a node or even an entire MySQL Cluster that is unable to restart due to there being insufficient memory space. Increasing these values should be acceptable, but it is recommended that such upgrades are performed in the same manner as a software upgrade, beginning with an update of the configuration file, and then restarting the management server followed by restarting each data node in turn.

Updates do not increase the amount of index memory used. Inserts take effect immediately; however, rows are not actually deleted until the transaction is committed.

Transaction parameters. The next three [NDBD] parameters that we discuss are important because they affect the number of parallel transactions and the sizes of transactions that can be handled by the system. `MaxNoOfConcurrentTransactions` sets the number of parallel transactions possible in a node. `MaxNoOfConcurrentOperations` sets the number of records that can be in update phase or locked simultaneously.

Both of these parameters (especially `MaxNoOfConcurrentOperations`) are likely targets for users setting specific values and not using the default value. The default value is set for systems using small transactions, to ensure that these do not use excessive memory.

- `MaxNoOfConcurrentTransactions`

For each active transaction in the cluster there must be a record in one of the cluster nodes. The task of coordinating transactions is spread among the nodes. The total number of transaction records in the cluster is the number of transactions in any given node times the number of nodes in the cluster.

Transaction records are allocated to individual MySQL servers. Normally, there is at least one transaction record allocated per connection that using any table in the cluster. For this reason, one should ensure that there are more transaction records in the cluster than there are concurrent connections to all MySQL servers in the cluster.

This parameter must be set to the same value for all cluster nodes.

Changing this parameter is never safe and doing so can cause a cluster to crash. When a node crashes, one of the nodes (actually the oldest surviving node) will build up the transaction state of all transactions ongoing in the crashed node at the time of the crash. It is thus important that this node has as many transaction records as the failed node.

The default value is 4096.

- `MaxNoOfConcurrentOperations`

It is a good idea to adjust the value of this parameter according to the size and number of transactions. When performing transactions of only a few operations each and not involving a great many records, there is no need to set this parameter very high. When performing large transactions involving many records need to set this parameter higher.

Records are kept for each transaction updating cluster data, both in the transaction coordinator and in the nodes where the actual updates are performed. These records contain state information needed to find UNDO records for rollback, lock queues, and other purposes.

This parameter should be set to the number of records to be updated simultaneously in transactions, divided by the number of cluster data nodes. For example, in a cluster which has four data nodes and which is expected to handle 1,000,000 concurrent updates using transactions, you should set this value to $1000000 / 4 = 250000$.

Read queries which set locks also cause operation records to be created. Some extra space is allocated within individual nodes to accommodate cases where the distribution is not perfect over the nodes.

When queries make use of the unique hash index, there are actually two operation records used per record in the transaction. The first record represents the read in the index table and the second handles the operation on the base table.

The default value is 32768.

This parameter actually handles two values that can be configured separately. The first of these specifies how many operation records are to be placed with the transaction coordinator. The second part specifies how many operation records are to be local to the database.

A very large transaction performed on an eight-node cluster requires as many operation records in the transaction coordinator as there are reads, updates, and deletes involved in the transaction. However, the operation records of the are spread over all eight nodes. Thus, if it is necessary to configure the system for one very large transaction, it is a good idea to configure the two parts separately. `MaxNoOfConcurrentOperations` will always be used to calculate the number of operation records in the transaction coordinator portion of the node.

It is also important to have an idea of the memory requirements for operation records. These consume about 1KB per record.

- **MaxNoOfLocalOperations**

By default, this parameter is calculated as $1.1 \times \text{MaxNoOfConcurrentOperations}$. This fits systems with many simultaneous transactions, none of them being very large. If there is a need to handle one very large transaction at a time and there are many nodes, it is a good idea to override the default value by explicitly specifying this parameter.

Transaction temporary storage. The next set of [NDBD] parameters is used to determine temporary storage when executing a statement that is part of a Cluster transaction. All records are released when the statement is completed and the cluster is waiting for the commit or rollback.

The default values for these parameters are adequate for most situations. However, users with a need to support transactions involving large numbers of rows or operations may need to increase these values to enable better parallelism in the system, whereas users whose applications require relatively small transactions can decrease the values to save memory.

- **MaxNoOfConcurrentIndexOperations**

For queries using a unique hash index, another temporary set of operation records is used during a query's execution phase. This parameter sets the size of that pool of records. Thus, this record is allocated only while executing a part of a query. As soon as this part has been executed, the record is released. The state needed to handle aborts and commits is handled by the normal operation records, where the pool size is set by the parameter **MaxNoOfConcurrentOperations**.

The default value of this parameter is 8192. Only in rare cases of extremely high parallelism using unique hash indexes should it be necessary to increase this value. Using a smaller value is possible and can save memory if the DBA is certain that a high degree of parallelism is not required for the cluster.

- **MaxNoOfFiredTriggers**

The default value of **MaxNoOfFiredTriggers** is 4000, which is sufficient for most situations. In some cases it can even be decreased if the DBA feels certain the need for parallelism in the cluster is not high.

A record is created when an operation is performed that affects a unique hash index. Inserting or deleting a record in a table with unique hash indexes or updating a column that is part of a unique hash index fires an insert or a delete in the index table. The resulting record is used to represent this index table operation while waiting for the original operation that fired it to complete. This operation is short-lived but can still require a large number of records in its pool for situations with many parallel write operations on a base table containing a set of unique hash indexes.

- **TransactionBufferMemory**

The memory affected by this parameter is used for tracking operations fired when updating index tables and reading unique indexes. This memory is used to store the key and column information for these operations. It is only very rarely that the value for this parameter needs to be altered from the default.

The default value for **TransactionBufferMemory** is 1MB.

Normal read and write operations use a similar buffer, whose usage is even more short-lived. The compile-time parameter **ZATTRINBUF_FILESIZE** (found in `ndb/src/kernel/blocks/Dbtc/Dbtc.hpp`) set to 4000×128 bytes (500KB). A similar buffer for key information, **ZDATABUF_FILESIZE** (also in `Dbtc.hpp`) contains $4000 \times 16 = 62.5$ KB of buffer space. **Dbtc** is the module that handles transaction coordination.

Scans and buffering. There are additional [NDBD] parameters in the **DbLqh** module (in `ndb/src/kernel/blocks/DbLqh/DbLqh.hpp`) that affect reads and updates. These include **ZATTRINBUF_FILESIZE**, set by default to 10000×128 bytes (1250KB) and **ZDATABUF_FILE_SIZE**, set by default to 10000×16 bytes (roughly 156KB) of buffer space. To date, there have been neither any reports from users nor any results from our own extensive tests suggesting that either of these compile-time limits should be increased.

- **MaxNoOfConcurrentScans**

This parameter is used to control the number of parallel scans that can be performed in the cluster. Each transaction coordinator can handle the number of parallel scans defined for this parameter. Each scan query is performed by scanning all partitions in parallel. Each partition scan uses a scan record in the node where the partition is located, the number of records being the value of this parameter times the number of nodes. The cluster should be able to sustain `MaxNoOfConcurrentScans` scans concurrently from all nodes in the cluster.

Scans are actually performed in two cases. The first of these cases occurs when no hash or ordered indexes exists to handle the query, in which case the query is executed by performing a full table scan. The second case is encountered when there is no hash index to support the query but there is an ordered index. Using the ordered index means executing a parallel range scan. The order is kept on the local partitions only, so it is necessary to perform the index scan on all partitions.

The default value of `MaxNoOfConcurrentScans` is 256. The maximum value is 500.

- **MaxNoOfLocalScans**
Specifies the number of local scan records if many scans are not fully parallelized. If the number of local scan records is not provided, it is calculated as the product of `MaxNoOfConcurrentScans` and the number of data nodes in the system. The minimum value is 32.
- **BatchSizePerLocalScan**
This parameter is used to calculate the number of lock records which must be there to handle many concurrent scan operations.
The default value is 64; this value has a strong connection to the `ScanBatchSize` defined in the SQL nodes.
- **LongMessageBuffer**
This is an internal buffer used for passing messages within individual nodes and between nodes. Although it is highly unlikely that this would need to be changed, it is configurable. By default, it is set to 1MB.

Memory Allocation

MaxAllocate

This is the maximum size of the memory unit to use when allocating memory for tables. In cases where `NDB` gives `OUT OF MEMORY` errors, but it is evident by examining the cluster logs or the output of `DUMP 1000` (see `DUMP 1000`) that all available memory has not yet been used, you can increase the value of this parameter (or `MaxNoOfTables`, or both) in order to cause `NDB` to make sufficient memory available.

This parameter was introduced in MySQL 5.1.20, and in MySQL Cluster 5.1 Carrier Grade Edition, in MySQL 5.1.15-ndb-6.1.12 and MySQL 5.1.19-ndb-6.2.3.

Logging and checkpointing. These `[NDBD]` parameters control log and checkpoint behavior.

- **NoOfFragmentLogFiles**
This parameter sets the number of REDO log files for the node, and thus the amount of space allocated to REDO logging. Because the REDO log files are organized in a ring, it is extremely important that the first and last log files in the set (sometimes referred to as the “head” and “tail” log files, respectively) do not meet. When these approach one another too closely, the node begins aborting all transactions encompassing updates due to a lack of room for new log records.

A REDO log record is not removed until three local checkpoints have been completed since that log record was inserted. Checkpointing frequency is determined by its own set of configuration parameters discussed elsewhere in this chapter.

How these parameters interact and proposals for how to configure them are discussed in [Section 16.4.6, “Configuring Parameters for Local Checkpoints”](#).

The default parameter value is 16, which means 16 sets of 4 16MB files for a total of 1024MB. In other words, REDO log space must be allocated in blocks of 64MB. In scenarios requiring a great many updates, the value for `NoOfFragmentLogFiles` may

need to be set as high as 300 or even higher to provide sufficient space for REDO logs.

If the checkpointing is slow and there are so many writes to the database that the log files are full and the log tail cannot be cut without jeopardizing recovery, all updating transactions are aborted with internal error code 410 (**Out of log file space temporarily**). This condition prevails until a checkpoint has completed and the log tail can be moved forward.

Important

This parameter cannot be changed “on the fly”; you must restart the node using `--initial`. If you wish to change this value for a running cluster, you can do so via a rolling node restart.

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

FragmentLogFileSize

Setting this parameter allows you to control directly the size of redo log files. The default is **16M**. This parameter was added in MySQL 5.1.15-ndb-6.1.11.

The following information applies to all MySQL Cluster users.

MaxNoOfOpenFiles

This parameter sets a ceiling on how many internal threads to allocate for open files. *Any situation requiring a change in this parameter should be reported as a bug.*

The default value is 40.

InitialNoOfOpenFiles

This parameter sets the initial number of internal threads to allocate for open files.

The default value is 27.

MaxNoOfSavedMessages

This parameter sets the maximum number of trace files that are kept before overwriting old ones. Trace files are generated when, for whatever reason, the node crashes.

The default is 25 trace files.

Metadata objects. The next set of **[NDBD]** parameters defines pool sizes for metadata objects, used to define the maximum number of attributes, tables, indexes, and trigger objects used by indexes, events, and replication between clusters. Note that these act merely as “suggestions” to the cluster, and any that are not specified revert to the default values shown.

MaxNoOfAttributes

Defines the number of attributes that can be defined in the cluster.

The default value is 1000, with the minimum possible value being 32. The maximum is 4294967039. Each attribute consumes around 200 bytes of storage per node due to the fact that all metadata is fully replicated on the servers.

When setting **MaxNoOfAttributes**, it is important to prepare in advance for any **ALTER TABLE** statements that you might want to perform in the future. This is due to the fact, during the execution of **ALTER TABLE** on a Cluster table, 3 times the number of attributes as in the original table are used. For example, if a table requires 100 attributes, and you want to be able to alter it later, you need to set the value of **MaxNoOfAttributes** to 300. Assuming that you can create all desired tables without any problems, a good rule of thumb is to add two times the number of attributes in the largest table to **MaxNoOfAttributes** to be sure. You

should also verify that this number is sufficient by trying an actual `ALTER TABLE` after configuring the parameter. If this is not successful, increase `MaxNoOfAttributes` by another multiple of the original value and test it again.

- `MaxNoOfTables`

A table object is allocated for each table, unique hash index, and ordered index. This parameter sets the maximum number of table objects for the cluster as a whole.

For each attribute that has a `BLOB` data type an extra table is used to store most of the `BLOB` data. These tables also must be taken into account when defining the total number of tables.

The default value of this parameter is 128. The minimum is 8 and the maximum is 1600. Each table object consumes approximately 20KB per node.

- `MaxNoOfOrderedIndexes`

For each ordered index in the cluster, an object is allocated describing what is being indexed and its storage segments. By default, each index so defined also defines an ordered index. Each unique index and primary key has both an ordered index and a hash index.

The default value of this parameter is 128. Each object consumes approximately 10KB of data per node.

- `MaxNoOfUniqueHashIndexes`

For each unique index that is not a primary key, a special table is allocated that maps the unique key to the primary key of the indexed table. By default, an ordered index is also defined for each unique index. To prevent this, you must specify the `USING HASH` option when defining the unique index.

The default value is 64. Each index consumes approximately 15KB per node.

- `MaxNoOfTriggers`

Internal update, insert, and delete triggers are allocated for each unique hash index. (This means that three triggers are created for each unique hash index.) However, an *ordered* index requires only a single trigger object. Backups also use three trigger objects for each normal table in the cluster.

Replication between clusters also makes use of internal triggers.

This parameter sets the maximum number of trigger objects in the cluster.

The default value is 768.

- `MaxNoOfIndexes`

This parameter is deprecated in MySQL 5.1; you should use `MaxNoOfOrderedIndexes` and `MaxNoOfUniqueHashIndexes` instead.

This parameter is used only by unique hash indexes. There needs to be one record in this pool for each unique hash index defined in the cluster.

The default value of this parameter is 128.

Boolean parameters. The behavior of data nodes is also affected by a set of `[NDBD]` parameters taking on boolean values. These parameters can each be specified as `TRUE` by setting them equal to `1` or `Y`, and as `FALSE` by setting them equal to `0` or `N`.

- `LockPagesInMainMemory`

For a number of operating systems, including Solaris and Linux, it is possible to lock a process into memory and so avoid any swapping to disk. This can be used to help guarantee the cluster's real-time characteristics.

Beginning with MySQL 5.1.15 (MySQL Cluster 5.1 Carrier Grade Edition: MySQL 5.1.15-ndb-6.1.1), this parameter takes one of the integer values `0`, `1`, or `2`, which act as follows:

- `0`: Disables locking. This is the default value.
- `1`: Performs the lock after allocating memory for the process.
- `2`: Performs the lock before memory for the process is allocated.

Previously, this parameter was a Boolean. `0` or `false` was the default setting, and disabled locking. `1` or `true` enabled locking of the process after its memory was allocated.

Important

Beginning with MySQL 5.1.15 (MySQL Cluster 5.1 Carrier Grade Edition: MySQL 5.1.15-ndb-6.1.1), it is no longer possible to use `true` or `false` for the value of this parameter; when upgrading from a previous version, you must change the value to `0`, `1`, or `2`.

StopOnError

This parameter specifies whether an `ndbd` process should exit or perform an automatic restart when an error condition is encountered.

This feature is enabled by default.

Diskless

It is possible to specify MySQL Cluster tables as *diskless*, meaning that tables are not checkpointed to disk and that no logging occurs. Such tables exist only in main memory. A consequence of using diskless tables is that neither the tables nor the records in those tables survive a crash. However, when operating in diskless mode, it is possible to run `ndbd` on a diskless computer.

Important

This feature causes the *entire* cluster to operate in diskless mode.

When this feature is enabled, Cluster online backup is disabled. In addition, a partial start of the cluster is not possible.

`Diskless` is disabled by default.

ODirect

Enabling this parameter causes `NDB` to attempt using `O_DIRECT` writes for LCP, backups, and redo logs, often lowering `kswapd` and CPU usage.

Warning

When using MySQL Cluster or MySQL Cluster 5.1 Carrier Grade Edition on Linux, `ODirect` should be enabled for 2.6 or newer kernels.

This parameter was added in the following releases:

- MySQL 5.1.20
- MySQL 5.1.15-ndb-6.1.11
- MySQL 5.1.19-ndb-6.2.3
- MySQL 5.1.19-ndb-6.3.0

`ODirect` is disabled by default.

RestartOnErrorInsert

This feature is accessible only when building the debug version where it is possible to insert errors in the execution of individual blocks of code as part of testing.

This feature is disabled by default.

Controlling Timeouts, Intervals, and Disk Paging

There are a number of [\[NDBD\]](#) parameters specifying timeouts and intervals between various actions in Cluster data nodes. Most of the timeout values are specified in milliseconds. Any exceptions to this are mentioned where applicable.

- ### TimeBetweenWatchDogCheck

To prevent the main thread from getting stuck in an endless loop at some point, a “watchdog” thread checks the main thread. This parameter specifies the number of milliseconds between checks. If the process remains in the same state after three checks, the watchdog thread terminates it.

This parameter can easily be changed for purposes of experimentation or to adapt to local conditions. It can be specified on a per-node basis although there seems to be little reason for doing so.

The default timeout is 6000 milliseconds (6 seconds).

- ### TimeBetweenWatchDogCheckInitial

This is similar to the [TimeBetweenWatchDogCheck](#) parameter, except that [TimeBetweenWatchDogCheckInitial](#) controls the amount of time that passes between execution checks inside a database node in the early start phases during which memory is allocated.

The default timeout is 6000 milliseconds (6 seconds).

This parameter was added in MySQL 5.1.20.

- ### StartPartialTimeout

This parameter specifies how long the Cluster waits for all data nodes to come up before the cluster initialization routine is invoked. This timeout is used to avoid a partial Cluster startup whenever possible.

The default value is 30000 milliseconds (30 seconds). 0 disables the timeout, in which case the cluster may start only if all nodes are available.

- ### StartPartitionedTimeout

If the cluster is ready to start after waiting for [StartPartialTimeout](#) milliseconds but is still possibly in a partitioned state, the cluster waits until this timeout has also passed.

The default timeout is 60000 milliseconds (60 seconds).

- ### StartFailureTimeout

If a data node has not completed its startup sequence within the time specified by this parameter, the node startup fails. Setting this parameter to 0 (the default value) means that no data node timeout is applied.

For nonzero values, this parameter is measured in milliseconds. For data nodes containing extremely large amounts of data, this parameter should be increased. For example, in the case of a data node containing several gigabytes of data, a period as long as 10–15 minutes (that is, 600000 to 1000000 milliseconds) might be required to perform a node restart.

- ### HeartbeatIntervalDbDb

One of the primary methods of discovering failed nodes is by the use of heartbeats. This parameter states how often heartbeat signals are sent and how often to expect to receive them. After missing three heartbeat intervals in a row, the node is declared dead. Thus, the maximum time for discovering a failure through the heartbeat mechanism is four times the heartbeat interval.

The default heartbeat interval is 1500 milliseconds (1.5 seconds). This parameter must not be changed drastically and should not vary widely between nodes. If one node uses 5000 milliseconds and the node watching it uses 1000 milliseconds, obviously the node will be declared dead very quickly. This parameter can be changed during an online software upgrade, but only in small increments.

- **HeartbeatIntervalDbApi**

Each data node sends heartbeat signals to each MySQL server (SQL node) to ensure that it remains in contact. If a MySQL server fails to send a heartbeat in time it is declared “dead,” in which case all ongoing transactions are completed and all resources released. The SQL node cannot reconnect until all activities initiated by the previous MySQL instance have been completed. The three-heartbeat criteria for this determination are the same as described for **HeartbeatIntervalDbDb**.

The default interval is 1500 milliseconds (1.5 seconds). This interval can vary between individual data nodes because each data node watches the MySQL servers connected to it, independently of all other data nodes.

- **TimeBetweenLocalCheckpoints**

This parameter is an exception in that it does not specify a time to wait before starting a new local checkpoint; rather, it is used to ensure that local checkpoints are not performed in a cluster where relatively few updates are taking place. In most clusters with high update rates, it is likely that a new local checkpoint is started immediately after the previous one has been completed.

The size of all write operations executed since the start of the previous local checkpoints is added. This parameter is also exceptional in that it is specified as the base-2 logarithm of the number of 4-byte words, so that the default value 20 means 4MB (4×2^{20}) of write operations, 21 would mean 8MB, and so on up to a maximum value of 31, which equates to 8GB of write operations.

All the write operations in the cluster are added together. Setting **TimeBetweenLocalCheckpoints** to 6 or less means that local checkpoints will be executed continuously without pause, independent of the cluster's workload.

- **TimeBetweenGlobalCheckpoints**

When a transaction is committed, it is committed in main memory in all nodes on which the data is mirrored. However, transaction log records are not flushed to disk as part of the commit. The reasoning behind this behavior is that having the transaction safely committed on at least two autonomous host machines should meet reasonable standards for durability.

It is also important to ensure that even the worst of cases — a complete crash of the cluster — is handled properly. To guarantee that this happens, all transactions taking place within a given interval are put into a global checkpoint, which can be thought of as a set of committed transactions that has been flushed to disk. In other words, as part of the commit process, a transaction is placed in a global checkpoint group. Later, this group's log records are flushed to disk, and then the entire group of transactions is safely committed to disk on all computers in the cluster.

This parameter defines the interval between global checkpoints. The default is 2000 milliseconds.

- **MySQL Cluster 5.1 Carrier Grade Edition**

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

TimeBetweenEpochs

This parameter defines the interval between synchronisation epochs for MySQL Cluster Replication. The default value is 100 milliseconds.

TimeBetweenEpochs is part of the implementation of “micro-GCPs”, which can be used to improve the performance of MySQL Cluster Replication. This parameter was introduced in MySQL-5.1.22-ndb-6.2.5 and MySQL 5.1.22-ndb-6.3.2.

The following information applies to all MySQL Cluster users.

- **TimeBetweenInactiveTransactionAbortCheck**

Timeout handling is performed by checking a timer on each transaction once for every interval specified by this parameter. Thus, if this parameter is set to 1000 milliseconds, every transaction will be checked for timing out once per second.

The default value is 1000 milliseconds (1 second).

- **TransactionInactiveTimeout**

This parameter states the maximum time that is permitted to lapse between operations in the same transaction before the transaction is aborted.

The default for this parameter is zero (no timeout). For a real-time database that needs to ensure that no transaction keeps locks for too long, this parameter should be set to a much smaller value. The unit is milliseconds.

- **TransactionDeadlockDetectionTimeout**

When a node executes a query involving a transaction, the node waits for the other nodes in the cluster to respond before continuing. A failure to respond can occur for any of the following reasons:

- The node is “dead”
- The operation has entered a lock queue
- The node requested to perform the action could be heavily overloaded.

This timeout parameter states how long the transaction coordinator waits for query execution by another node before aborting the transaction, and is important for both node failure handling and deadlock detection. In MySQL 5.1.10 and earlier versions, setting it too high could cause undesirable behavior in situations involving deadlocks and node failure. Beginning with MySQL 5.1.11, active transactions occurring during node failures are actively aborted by the Cluster Transaction Coordinator, and so high settings are no longer an issue with this parameter.

The default timeout value is 1200 milliseconds (1.2 seconds).

- **DiskSyncSize**

This is the maximum number of bytes to store before flushing data to a local checkpoint file.

The default value is 4M (4 megabytes).

This parameter was added in MySQL 5.1.12.

- **DiskCheckpointSpeed**

The amount of data, in bytes per second, that is sent to disk during a local checkpoint.

The default value is 10M (10 megabytes per second).

This parameter was added in MySQL 5.1.12.

- **DiskCheckpointSpeedInRestart**

The amount of data, in bytes per second, that is sent to disk during a local checkpoint as part of a restart operation.

The default value is 100M (100 megabytes per second).

This parameter was added in MySQL 5.1.12.

- **NoOfDiskPagesToDiskAfterRestartTUP**

When executing a local checkpoint, the algorithm flushes all data pages to disk. Merely doing so as quickly as possible without any moderation is likely to impose excessive loads on processors, networks, and disks. To control the write speed, this parameter specifies how many pages per 100 milliseconds are to be written. In this context, a “page” is defined as 8KB. This parameter is specified in units of 80KB per second, so setting `NoOfDiskPagesToDiskAfterRestartTUP` to a value of `20` entails writing 1.6MB in data pages to disk each second during a local checkpoint. This value includes the writing of UNDO log records for data pages. That is, this parameter handles the limitation of writes from data memory. (See the entry for `IndexMemory` for information about index pages.)

In short, this parameter specifies how quickly to execute local checkpoints. It operates in conjunction with `NoOfFragmentLogFiles`, `DataMemory`, and `IndexMemory`.

For more information about the interaction between these parameters and possible strategies for choosing appropriate values for them, see [Section 16.4.6, “Configuring Parameters for Local Checkpoints”](#).

The default value is 40 (3.2MB of data pages per second).

Note

This parameter is deprecated as of MySQL 5.1.6. For MySQL 5.1.12 and later versions, use `DiskCheckpointSpeed` and `DiskSyncSize` instead.

- `NoOfDiskPagesToDiskAfterRestartACC`

This parameter uses the same units as `NoOfDiskPagesToDiskAfterRestartTUP` and acts in a similar fashion, but limits the speed of writing index pages from index memory.

The default value of this parameter is 20 (1.6MB of index memory pages per second).

Note

This parameter is deprecated as of MySQL 5.1.6. For MySQL 5.1.12 and later versions, use `DiskCheckpointSpeed` and `DiskSyncSize`.

- `NoOfDiskPagesToDiskDuringRestartTUP`

This parameter is used in a fashion similar to `NoOfDiskPagesToDiskAfterRestartTUP` and `NoOfDiskPagesToDiskAfterRestartACC`, only it does so with regard to local checkpoints executed in the node when a node is restarting. A local checkpoint is always performed as part of all node restarts. During a node restart it is possible to write to disk at a higher speed than at other times, because fewer activities are being performed in the node.

This parameter covers pages written from data memory.

The default value is 40 (3.2MB per second).

Note

This parameter is deprecated as of MySQL 5.1.6. For MySQL 5.1.12 and later versions, use `DiskCheckpointSpeedInRestart` and `DiskSyncSize`.

- `NoOfDiskPagesToDiskDuringRestartACC`

Controls the number of index memory pages that can be written to disk during the local checkpoint phase of a node restart.

As with `NoOfDiskPagesToDiskAfterRestartTUP` and `NoOfDiskPagesToDiskAfterRestartACC`, values for this parameter are expressed in terms of 8KB pages written per 100 milliseconds (80KB/second).

The default value is 20 (1.6MB per second).

Note

This parameter is deprecated as of MySQL 5.1.6. For MySQL 5.1.12 and later versions, use `DiskCheckpointSpeedInRestart` and `DiskSyncSize`.

- **ArbitrationTimeout**

This parameter specifies how long data nodes wait for a response from the arbitrator to an arbitration message. If this is exceeded, the network is assumed to have split.

The default value is 1000 milliseconds (1 second).

Buffering and logging. Several [\[NDBD\]](#) configuration parameters enable the advanced user to have more control over the resources used by node processes and to adjust various buffer sizes at need.

These buffers are used as front ends to the file system when writing log records to disk. If the node is running in diskless mode, these parameters can be set to their minimum values without penalty due to the fact that disk writes are “faked” by the [NDB](#) storage engine's filesystem abstraction layer.

- **UndoIndexBuffer**

The UNDO index buffer, whose size is set by this parameter, is used during local checkpoints. The [NDB](#) storage engine uses a recovery scheme based on checkpoint consistency in conjunction with an operational REDO log. To produce a consistent checkpoint without blocking the entire system for writes, UNDO logging is done while performing the local checkpoint. UNDO logging is activated on a single table fragment at a time. This optimization is possible because tables are stored entirely in main memory.

The UNDO index buffer is used for the updates on the primary key hash index. Inserts and deletes rearrange the hash index; the [NDB](#) storage engine writes UNDO log records that map all physical changes to an index page so that they can be undone at system restart. It also logs all active insert operations for each fragment at the start of a local checkpoint.

Reads and updates set lock bits and update a header in the hash index entry. These changes are handled by the page-writing algorithm to ensure that these operations need no UNDO logging.

This buffer is 2MB by default. The minimum value is 1MB, which is sufficient for most applications. For applications doing extremely large or numerous inserts and deletes together with large transactions and large primary keys, it may be necessary to increase the size of this buffer. If this buffer is too small, the [NDB](#) storage engine issues internal error code 677 ([Index UNDO buffers overloaded](#)).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

- **UndoDataBuffer**

This parameter sets the size of the UNDO data buffer, which performs a function similar to that of the UNDO index buffer, except the UNDO data buffer is used with regard to data memory rather than index memory. This buffer is used during the local checkpoint phase of a fragment for inserts, deletes, and updates.

Because UNDO log entries tend to grow larger as more operations are logged, this buffer is also larger than its index memory counterpart, with a default value of 16MB.

This amount of memory may be unnecessarily large for some applications. In such cases, it is possible to decrease this size to a minimum of 1MB.

It is rarely necessary to increase the size of this buffer. If there is such a need, it is a good idea to check whether the disks can actually handle the load caused by database update activity. A lack of sufficient disk space cannot be overcome by increasing the size of this buffer.

If this buffer is too small and gets congested, the [NDB](#) storage engine issues internal error code 891 ([DATA UNDO BUFFERS OVERLOADED](#)).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

RedoBuffer

All update activities also need to be logged. The REDO log makes it possible to replay these updates whenever the system is restarted. The NDB recovery algorithm uses a “fuzzy” checkpoint of the data together with the UNDO log, and then applies the REDO log to play back all changes up to the restoration point.

`RedoBuffer` sets the size of the buffer in which the REDO log is written, and is 8MB by default. The minimum value is 1MB.

If this buffer is too small, the NDB storage engine issues error code 1221 (`REDO log buffers overloaded`).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

Controlling log messages. In managing the cluster, it is very important to be able to control the number of log messages sent for various event types to `stdout`. For each event category, there are 16 possible event levels (numbered 0 through 15). Setting event reporting for a given event category to level 15 means all event reports in that category are sent to `stdout`; setting it to 0 means that there will be no event reports made in that category.

By default, only the startup message is sent to `stdout`, with the remaining event reporting level defaults being set to 0. The reason for this is that these messages are also sent to the management server's cluster log.

An analogous set of levels can be set for the management client to determine which event levels to record in the cluster log.

- `LogLevelStartup`
The reporting level for events generated during startup of the process.
The default level is 1.
- `LogLevelShutdown`
The reporting level for events generated as part of graceful shutdown of a node.
The default level is 0.
- `LogLevelStatistic`
The reporting level for statistical events such as number of primary key reads, number of updates, number of inserts, information relating to buffer usage, and so on.
The default level is 0.
- `LogLevelCheckpoint`
The reporting level for events generated by local and global checkpoints.
The default level is 0.
- `LogLevelNodeRestart`
The reporting level for events generated during node restart.
The default level is 0.
- `LogLevelConnection`
The reporting level for events generated by connections between cluster nodes.

The default level is 0.

- **LogLevelError**

The reporting level for events generated by errors and warnings by the cluster as a whole. These errors do not cause any node failure but are still considered worth reporting.

The default level is 0.

- **LogLevelCongestion**

The reporting level for events generated by congestion. These errors do not cause node failure but are still considered worth reporting.

The default level is 0.

- **LogLevelInfo**

The reporting level for events generated for information about the general state of the cluster.

The default level is 0.

- **MemReportFrequency**

This parameter controls how often data node memory usage reports are recorded in the cluster log; it is an integer value representing the number of seconds between reports.

Each data node's data memory and index memory usage is logged as both a percentage and a number of 32 KB pages of the **DataMemory** and **IndexMemory**, respectively, set in the **config.ini** file. For example, if **DataMemory** is equal to 100 MB, and a given data node is using 50 MB for data memory storage, the corresponding line in the cluster log might look like this:

```
2006-12-24 01:18:16 [MgmSrvr] INFO -- Node 2: Data usage is 50%(1280 32K pages of total 2560)
```

MemReportFrequency is not a required parameter. If used, it can be set for all cluster data nodes in the **[NDBD DEFAULT]** section of **config.ini**, and can also be set or overridden for individual data nodes in the corresponding **[NDBD]** sections of the configuration file. The minimum value — which is also the default value — is 0, in which case memory reports are logged only when memory usage reaches certain percentages (80%, 90%, and 100%), as mentioned in the discussion of statistics events in [Section 16.7.3.2, “Log Events”](#).

This parameter was added in MySQL Cluster 5.1.16. (MySQL Cluster 5.1 Carrier Grade Edition: MySQL Cluster 5.1.14-ndb-6.1.0).

Backup parameters. The **[NDBD]** parameters discussed in this section define memory buffers set aside for execution of online backups.

- **BackupDataBufferSize**

In creating a backup, there are two buffers used for sending data to the disk. The backup data buffer is used to fill in data recorded by scanning a node's tables. Once this buffer has been filled to the level specified as **BackupWriteSize** (see below), the pages are sent to disk. While flushing data to disk, the backup process can continue filling this buffer until it runs out of space. When this happens, the backup process pauses the scan and waits until some disk writes have completed freed up memory so that scanning may continue.

The default value is 2MB.

- **BackupLogBufferSize**

The backup log buffer fulfills a role similar to that played by the backup data buffer, except that it is used for generating a log of all

table writes made during execution of the backup. The same principles apply for writing these pages as with the backup data buffer, except that when there is no more space in the backup log buffer, the backup fails. For that reason, the size of the backup log buffer must be large enough to handle the load caused by write activities while the backup is being made. See [Section 16.8.4, “Configuration for Cluster Backup”](#).

The default value for this parameter should be sufficient for most applications. In fact, it is more likely for a backup failure to be caused by insufficient disk write speed than it is for the backup log buffer to become full. If the disk subsystem is not configured for the write load caused by applications, the cluster is unlikely to be able to perform the desired operations.

It is preferable to configure cluster nodes in such a manner that the processor becomes the bottleneck rather than the disks or the network connections.

The default value is 2MB.

- **BackupMemory**

This parameter is simply the sum of [BackupDataBufferSize](#) and [BackupLogBufferSize](#).

The default value is 2MB + 2MB = 4MB.

Important

If [BackupDataBufferSize](#) and [BackupLogBufferSize](#) taken together exceed 4MB, then this parameter must be set explicitly in the `config.ini` file to their sum.

- **MySQL Cluster 5.1 Carrier Grade Edition**

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#).

This parameter controls how often backup status reports are issued in the management client during a backup, as well as how often such reports are written to the cluster log (provided cluster event logging is configured to allow it — see [Section 16.4.4.5, “Defining Data Nodes”](#) []). [BackupReportFrequency](#) represents the time in seconds between backup status reports.

The default value is 0.

This parameter was added in MySQL 5.1.19-ndb-6.2.3.

The following information applies to all MySQL Cluster users.

- **BackupWriteSize**

This parameter specifies the default size of messages written to disk by the backup log and backup data buffers.

The default value is 32KB.

- **BackupMaxWriteSize**

This parameter specifies the maximum size of messages written to disk by the backup log and backup data buffers.

The default value is 256KB.

Important

When specifying these parameters, the following relationships must hold true. Otherwise, the data node will be unable to start.

- [BackupDataBufferSize](#) >= [BackupWriteSize](#) + 188KB

- `BackupLogBufferSize >= BackupWriteSize + 16KB`
- `BackupMaxWriteSize >= BackupWriteSize`

16.4.4.6. Defining SQL and Other API Nodes

The `[MYSQLD]` and `[API]` sections in the `config.ini` file define the behavior of the MySQL servers (SQL nodes) and other applications (API nodes) used to access cluster data. None of the parameters shown is required. If no computer or host name is provided, any host can use this SQL or API node.

Generally speaking, a `[MYSQLD]` section is used to indicate a MySQL server providing an SQL interface to the cluster, and an `[API]` section is used for applications other than `mysqld` processes accessing cluster data, but the two designations are actually synonymous; you can, for instance, list parameters for a MySQL server acting as an SQL node in an `[API]` section.

- **Id**

The **Id** value is used to identify the node in all cluster internal messages. It must be an integer in the range 1 to 63 inclusive, and must be unique among all node IDs within the cluster.

- **ExecuteOnComputer**

This refers to the **Id** set for one of the computers (hosts) defined in a `[COMPUTER]` section of the configuration file.

- **HostName**

Specifying this parameter defines the hostname of the computer on which the SQL node (API node) is to reside. To specify a host-name other than `localhost`, either this parameter or **ExecuteOnComputer** is required.

- **ArbitrationRank**

This parameter defines which nodes can act as arbitrators. Both MGM nodes and SQL nodes can be arbitrators. A value of 0 means that the given node is never used as an arbitrator, a value of 1 gives the node high priority as an arbitrator, and a value of 2 gives it low priority. A normal configuration uses the management server as arbitrator, setting its **ArbitrationRank** to 1 (the default) and those for all SQL nodes to 0.

Beginning with MySQL 5.1.16 (*MySQL Cluster 5.1 Carrier Grade Edition: MySQL 5.1.15-ndb-6.1.3*), it is possible to disable arbitration completely by setting **ArbitrationRank** to 0 on all management and SQL nodes.

- **ArbitrationDelay**

Setting this parameter to any other value than 0 (the default) means that responses by the arbitrator to arbitration requests will be delayed by the stated number of milliseconds. It is usually not necessary to change this value.

- **BatchByteSize**

For queries that are translated into full table scans or range scans on indexes, it is important for best performance to fetch records in properly sized batches. It is possible to set the proper size both in terms of number of records (**BatchSize**) and in terms of bytes (**BatchByteSize**). The actual batch size is limited by both parameters.

The speed at which queries are performed can vary by more than 40% depending upon how this parameter is set. In future releases, MySQL Server will make educated guesses on how to set parameters relating to batch size, based on the query type.

This parameter is measured in bytes and by default is equal to 32KB.

- **BatchSize**

This parameter is measured in number of records and is by default set to 64. The maximum size is 992.

- **MaxScanBatchSize**

The batch size is the size of each batch sent from each data node. Most scans are performed in parallel to protect the MySQL Server from receiving too much data from many nodes in parallel; this parameter sets a limit to the total batch size over all nodes.

The default value of this parameter is set to 256KB. Its maximum size is 16MB.

You can obtain some information from a MySQL server running as a Cluster SQL node using `SHOW STATUS` in the `mysql` client, as shown here:

```
mysql> SHOW STATUS LIKE 'ndb%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_cluster_node_id | 5 |
| Ndb_config_from_host | 192.168.0.112 |
| Ndb_config_from_port | 1186 |
| Ndb_number_of_storage_nodes | 4 |
+-----+-----+
4 rows in set (0.02 sec)
```

For information about these Cluster system status variables, see [Section 5.2.5, “Status Variables”](#).

16.4.4.7. Cluster TCP/IP Connections

TCP/IP is the default transport mechanism for establishing connections in MySQL Cluster. It is normally not necessary to define connections because Cluster automatically set ups a connection between each of the data nodes, between each data node and all MySQL server nodes, and between each data node and the management server. (For one exception to this rule, see [Section 16.4.4.8, “TCP/IP Connections Using Direct Connections”](#).) `[TCP]` sections in the `config.ini` file explicitly define TCP/IP connections between nodes in the cluster.

It is necessary to define a connection only to override the default connection parameters. In that case, it is necessary to define at least `NodeId1`, `NodeId2`, and the parameters to change.

Important

Any `[TCP]` sections in the `config.ini` file should be listed last, following any other sections in the file. This is not required for a `[TCP DEFAULT]` section. This is a known issue with the way in which the `config.ini` file is read by the cluster management server.

It is also possible to change the default values for these parameters by setting them in the `[TCP DEFAULT]` section.

- **NodeId1, NodeId2**

To identify a connection between two nodes it is necessary to provide their node IDs in the `[TCP]` section of the configuration file. These are the same unique `Id` values for each of these nodes as described in [Section 16.4.4.6, “Defining SQL and Other API Nodes”](#).

- **SendBufferMemory**

TCP transporters use a buffer to store all messages before performing the send call to the operating system. When this buffer reaches 64KB its contents are sent; these are also sent when a round of messages have been executed. To handle temporary overload situations it is also possible to define a bigger send buffer. The default size of the send buffer is 256 KB. The minimum size is 64 KB; the theoretical maximum is 4 GB.

- **SendSignalId**

To be able to retrace a distributed message datagram, it is necessary to identify each message. When this parameter is set to `Y`, message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- **Checksum**

This parameter is a boolean parameter (enabled by setting it to **Y** or **1**, disabled by setting it to **N** or **0**). It is disabled by default. When it is enabled, checksums for all messages are calculated before they placed in the send buffer. This feature ensures that messages are not corrupted while waiting in the send buffer, or by the transport mechanism.

- **PortNumber** (*OBSOLETE*)

This formerly specified the port number to be used for listening for connections from other nodes. This parameter should no longer be used.

- **ReceiveBufferMemory**

Specifies the size of the buffer used when receiving data from the TCP/IP socket. There is seldom any need to change this parameter from its default value of 64 KB, except possibly to save memory. The minimum possible value is 16K; the theoretical maximum is 4G.

16.4.4.8. TCP/IP Connections Using Direct Connections

Setting up a cluster using direct connections between data nodes requires specifying explicitly the crossover IP addresses of the data nodes so connected in the **[TCP]** section of the cluster **config.ini** file.

In the following example, we envision a cluster with at least four hosts, one each for a management server, an SQL node, and two data nodes. The cluster as a whole resides on the **172.23.72.*** subnet of a LAN. In addition to the usual network connections, the two data nodes are connected directly using a standard crossover cable, and communicate with one another directly using IP addresses in the **1.1.0.*** address range as shown:

```
# Management Server
[NDB_MGMD]
Id=1
HostName=172.23.72.20

# SQL Node
[MYSQLD]
Id=2
HostName=172.23.72.21

# Data Nodes
[NDBD]
Id=3
HostName=172.23.72.22

[NDBD]
Id=4
HostName=172.23.72.23

# TCP/IP Connections
[TCP]
NodeId1=3
NodeId2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

The **HostNameN** parameter, where **N** is an integer, is used only when specifying direct TCP/IP connections.

The use of direct connections between data nodes can improve the cluster's overall efficiency by allowing the data nodes to bypass an Ethernet device such as a switch, hub, or router, thus cutting down on the cluster's latency. It is important to note that to take the best advantage of direct connections in this fashion with more than two data nodes, you must have a direct connection between each data node and every other data node in the same node group.

16.4.4.9. Shared-Memory Connections

MySQL Cluster attempts to use the shared memory transporter and configure it automatically where possible. **[SHM]** sections in the **config.ini** file explicitly define shared-memory connections between nodes in the cluster. When explicitly defining shared memory as the connection method, it is necessary to define at least **NodeId1**, **NodeId2** and **ShmKey**. All other parameters have default values that should work well in most cases.

■ Important

SHM functionality is considered experimental only. It is not officially supported in any MySQL release series up to and including 5.1. This means that you must determine for yourself or by using our free resources (forums, mailing lists) whether it can be made to work correctly in your specific case.

- **NodeId1, NodeId2**
To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as **NodeId1** and **NodeId2**.
- **ShmKey**
When setting up shared memory segments, a node ID, expressed as an integer, is used to identify uniquely the shared memory segment to use for the communication. There is no default value.
- **ShmSize**
Each SHM connection has a shared memory segment where messages between nodes are placed by the sender and read by the reader. The size of this segment is defined by **ShmSize**. The default value is 1MB.
- **SendSignalId**
To retrace the path of a distributed message, it is necessary to provide each message with a unique identifier. Setting this parameter to **Y** causes these message IDs to be transported over the network as well. This feature is disabled by default in production builds, and enabled in **-debug** builds.
- **Checksum**
This parameter is a boolean (**Y/N**) parameter which is disabled by default. When it is enabled, checksums for all messages are calculated before being placed in the send buffer.

This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

16.4.4.10. SCI Transport Connections

[**SCI**] sections in the **config.ini** file explicitly define SCI (Scalable Coherent Interface) connections between cluster nodes. Using SCI transporters in MySQL Cluster is supported only when the MySQL binaries are built using **--with-ndb-sci=/your/path/to/SCI**. The **path** should point to a directory that contains at a minimum **lib** and **include** directories containing SISI libraries and header files. (See [Section 16.12, “Using High-Speed Interconnects with MySQL Cluster”](#) for more information about SCI.)

In addition, SCI requires specialized hardware.

It is strongly recommended to use SCI Transporters only for communication between **ndbd** processes. Note also that using SCI Transporters means that the **ndbd** processes never sleep. For this reason, SCI Transporters should be used only on machines having at least two CPUs dedicated for use by **ndbd** processes. There should be at least one CPU per **ndbd** process, with at least one CPU left in reserve to handle operating system activities.

- **NodeId1, NodeId2**
To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as **NodeId1** and **NodeId2**.
- **Host1SciId0**
This identifies the SCI node ID on the first Cluster node (identified by **NodeId1**).

- [Host1SciId1](#)

It is possible to set up SCI Transporters for failover between two SCI cards which then should use separate networks between the nodes. This identifies the node ID and the second SCI card to be used on the first node.

- [Host2SciId0](#)

This identifies the SCI node ID on the second Cluster node (identified by [NodeId2](#)).

- [Host2SciId1](#)

When using two SCI cards to provide failover, this parameter identifies the second SCI card to be used on the second node.

- [SharedBufferSize](#)

Each SCI transporter has a shared memory segment used for communication between the two nodes. Setting the size of this segment to the default value of 1MB should be sufficient for most applications. Using a smaller value can lead to problems when performing many parallel inserts; if the shared buffer is too small, this can also result in a crash of the [ndbd](#) process.

- [SendLimit](#)

A small buffer in front of the SCI media stores messages before transmitting them over the SCI network. By default, this is set to 8KB. Our benchmarks show that performance is best at 64KB but 16KB reaches within a few percent of this, and there was little if any advantage to increasing it beyond 8KB.

- [SendSignalId](#)

To trace a distributed message it is necessary to identify each message uniquely. When this parameter is set to [Y](#), message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in [-debug](#) builds.

- [Checksum](#)

This parameter is a boolean value, and is disabled by default. When [Checksum](#) is enabled, checksums are calculated for all messages before they are placed in the send buffer. This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

16.4.5. Overview of Cluster Configuration Parameters

The next three sections provide summary tables of MySQL Cluster configuration parameters used in the [config.ini](#) file to govern the cluster's functioning. Each table lists the parameters for one of the Cluster node process types ([ndbd](#), [ndb_mgmd](#), and [mysqld](#)), and includes the parameter's type as well as its default, minimum, and maximum values as applicable.

It is also stated what type of restart is required (node restart or system restart) — and whether the restart must be done with [-initial](#) — to change the value of a given configuration parameter. This information is provided in each table's **Restart Type** column, which contains one of the values shown in this list:

- [N](#): Node Restart
- [IN](#): Initial Node Restart
- [S](#): System Restart
- [IS](#): Initial System Restart

When performing a node restart or an initial node restart, all of the cluster's data nodes must be restarted in turn (also referred to as a *rolling restart*). It is possible to update cluster configuration parameters marked [N](#) or [IN](#) online — that is, without shutting down the cluster — in this fashion. An initial node restart requires restarting each [ndbd](#) process with the [--initial](#) option.

A system restart requires a complete shutdown and restart of the entire cluster. An initial system restart requires taking a backup of the

cluster, wiping the cluster filesystem after shutdown, and then restoring from the backup following the restart.

In any cluster restart, all of the cluster's management servers must be restarted in order for them to read the updated configuration parameter values.

Important

Values for numeric cluster parameters can generally be increased without any problems, although it is advisable to do so progressively, making such adjustments in relatively small increments. However, decreasing the values of such parameters — particularly those relating to memory usage and disk space — is not to be undertaken lightly, and it is recommended that you do so only following careful planning and testing. In addition, it is the generally the case that parameters relating to memory and disk usage which can be raised using a simple node restart require an initial node restart to be lowered.

Because some of these parameters can be used for configuring more than one type of cluster node, they may appear in more than one of the tables.

(Note that [4294967039](#) — which often appears as a maximum value in these tables — is equal to $2^{32} - 2^8 - 1$.)

16.4.5.1. Data Node Configuration Parameters

The following table provides information about parameters used in the `[NDBD]` or `[NDB_DEFAULT]` sections of a `config.ini` file for configuring MySQL Cluster data nodes. For detailed descriptions and other additional information about each of these parameters, see [Section 16.4.4.5, “Defining Data Nodes”](#).

Restart Type Column Values

- **N**: Node Restart
- **IN**: Initial Node Restart
- **S**: System Restart
- **IS**: Initial System Restart

See [Section 16.4.5, “Overview of Cluster Configuration Parameters”](#), for additional explanations of these abbreviations.

Parameter Name	Type / Unit	Default Value	Minimum Value	Maximum Value	Restart Type
ArbitrationTimeout	milliseconds	3000	10	4294967039	N
BackupDataBufferSize	bytes	2M	0	4294967039	N
BackupDataDir	string	FileSystemPath/BACKUP	N/A	N/A	IN
BackupLogBufferSize	bytes	2M	0	4294967039	N
BackupMemory	bytes	4M	0	4294967039	N
BackupReportFrequency (Added in MySQL 5.1.19-ndb-6.2.3)	seconds	0	0	4294967039	N
BackupWriteSize	bytes	32K	2K	4294967039	N
BackupMaxWriteSize	bytes	256K	2K	4294967039	N
BatchSizePerLocalScan	integer	64	1	992	N
DataDir	string	/var/lib/mysql-cluster	N/A	N/A	IN
DataMemory	bytes	80M	1M	1024G (subject to avail-	N

				able system RAM and size of IndexMemory)	
DiskCheckpointSpeed (added in MySQL 5.1.12)	integer (number of bytes per second)	10M	1M	4294967039	N
DiskCheckpointSpeedInRestart (added in MySQL 5.1.12)	integer (number of bytes per second)	100M	1M	4294967039	N
Diskless	true false (1 0)	0	0	1	IS
DiskPageBufferMemory	bytes	64M	4M	1024G	IS
DiskSyncSize (added in MySQL 5.1.12)	integer (number of bytes)	4M	32K	4294967039	N
ExecuteOnComputer	integer				
FilePath (Added in MySQL-5.1.15-ndb-6.1.11)	string	value specified for DataDir	N/A	N/A	IN
FragmentLogFileSize	integer	16M	4M	1G	N
HeartbeatIntervalDbApi	milliseconds	1500	100	4294967039	N
HeartbeatIntervalDbDb	milliseconds	1500	10	4294967039	N
HostName	string	localhost	N/A	N/A	S
Id	integer	<i>None</i>	1	63	N
IndexMemory	bytes	18M	1M	1024G (subject to available system RAM and size of DataMemory)	N
InitialNoOfOpenFiles	integer	27	20	4294967039	N
LockPagesInMainMemory	As of MySQL 5.1.15 (MySQL	0	0	1	N

	<i>QL Cluster 5.1 Carrier Grade Edition: MySQL QL 5.1.1 5-nd b- 6.1.1); integer ; pre- vi-</i>				

	: true false (1 0)				
LogLevelCheckpoint	in- teger	0	0	15	IN
LogLevelCongestion	in- teger	0	0	15	N
LogLevelConnection	in- teger	0	0	15	N
LogLevelError	in- teger	0	0	15	N
LogLevelInfo	in- teger	0	0	15	N
LogLevelNodeRestart	in- teger	0	0	15	N
LogLevelShutdown	in- teger	0	0	15	N
LogLevelStartup	in- teger	1	0	15	N
LogLevelStatistic	in- teger	0	0	15	N
LongMessageBuffer	bytes	1M	512K	4294967039	N
MaxAllocate (Added in MySQL 5.1.15-ndb-6.1.12 and MySQL 5.1.19-ndb-6.2.3)	in- teger	32M	1M	1G	N
MaxNoOfAttributes	in- teger	1000	32	4294967039	N
MaxNoOfConcurrentIndexOperations	in- teger	8K	0	4294967039	N
MaxNoOfConcurrentOperations	in- teger	32768	32	4294967039	N
MaxNoOfConcurrentScans	in- teger	256	2	500	N
MaxNoOfConcurrentTransactions	in- teger	4096	32	4294967039	N
MaxNoOfFiredTriggers	in- teger	4000	0	4294967039	N
MaxNoOfIndexes (DEPRECATED — use MaxNoOfOrderedIndexes or MaxNoOfUniqueHashIndexes instead)	in- teger	128	0	4294967039	N
MaxNoOfLocalOperations	in- teger	UNDEFINED	32	4294967039	N
MaxNoOfLocalScans	in- teger	UNDEFINED (see de- scription)	32	4294967039	N
MaxNoOfOpenFiles	in- teger	40	20	4294967039	N
MaxNoOfOrderedIndexes	in- teger	128	0	4294967039	N
MaxNoOfSavedMessages	in- teger	25	0	4294967039	N
MaxNoOfTables	in- teger	128	8	4294967039	N

MaxNoOfTriggers	in- teger	768	0	4294967039	N
MaxNoOfUniqueHashIndexes	in- teger	64	0	4294967039	N
MemReportFrequency (added in MySQL 5.1.16; MySQL Cluster 5.1 Carrier Grade Edition: added in MySQL 5.1.14-ndb-6.1.0)	in- teger	0	0	4294967039	N
NoOfDiskPagesToDiskAfter-RestartACC (<i>DEPRECATED</i> as of MySQL 5.1.6)	in- teger (num ber of 8KB page s per 100 milli- seco nds)	20 (= 20 * 80KB = 1.6MB/second)	1	4294967039	N
NoOfDiskPagesToDiskAfter-RestartTUP (<i>DEPRECATED</i> as of MySQL 5.1.6)	in- teger (num ber of 8KB page s per 100 milli- seco nds)	40 (= 40 * 80KB = 3.2MB/second)	1	4294967039	N
NoOfDiskPagesToDiskDuringRestartACC (<i>DEPRECATED</i> as of MySQL 5.1.6)	in- teger (num ber of 8KB page s per 100 milli- seco nds)	20 (= 20 * 80KB = 1.6MB/second)	1	4294967039	N
NoOfDiskPagesToDiskDuringRestartTUP (<i>DEPRECATED</i> as of MySQL 5.1.6)	in- teger (num ber of 8KB page s per 100 milli- seco nds)	40 (= 40 * 80KB = 3.2MB/second)	1	4294967039	N
NoOfFragmentLogFiles	in- teger	16	3	4294967039	IN
NoOfReplicas	in- teger	<i>None</i>	1	4	IS

ODirect	boolean	0	0	1	NR
RedoBuffer	bytes	8M	1M	4294967039	N
RestartOnErrorInsert (<i>DE-BUG BUILDS ONLY</i>)	true false (1 0)	0	0	1	N
ServerPort (<i>OBSOLETE</i>)	integer	1186	0	4294967039	N
SharedGlobalmemory	bytes	20M	0	65536G	N
StartFailureTimeout	milliseconds	0	0	4294967039	N
StartPartialTimeout	milliseconds	30000	0	4294967039	N
StartPartitionedTimeout	milliseconds	60000	0	4294967039	N
StopOnError	true false (1 0)	1	0	1	N
StringMemory	integer or percentage (see description for details)	0	0	4294967039	S
TcpBind_INADDR_ANY (<i>MySQL Cluster 5.1 Carrier Grade Edition only: added in MySQL 5.1.16-ndb-6.2.0</i>)	true false (1 0)	1	0	0	N
TimeBetweenEpochs (<i>MySQL Cluster 5.1 Carrier Grade Edition only: added in MySQL 5.1.22-ndb-6.2.5 and MySQL 5.1.22-ndb-6.3.2</i>)	milliseconds	100	0	32000	N
TimeBetweenGlobalCheckpoints	milliseconds	2000	10	32000	N
TimeBetweenInactiveTransactionAbortCheck	milliseconds	1000	1000	4294967039	N
TimeBetweenLocalCheckpoints	integer (number of 4-byte words as a	20 (= $4 * 2^{20}$ = 4MB write operations)	0	31	N

	base-2 log-arithm)				
TimeBetweenWatchDogCheck	milliseconds	6000	70	4294967039	N
TimeBetweenWatchDogCheckInitial (added in MySQL 5.1.20)	milliseconds	6000	70	4294967039	N
TransactionBufferMemory	bytes	1M	1K	4294967039	N
TransactionDeadlockDetectionTimeout	milliseconds	1200	50	4294967039	N
TransactionInactiveTimeout	milliseconds	0	0	4294967039	N
UndoDataBuffer (<i>OBSOLETE</i>)	bytes	16M	1M	4294967039	N
UndoIndexBuffer (<i>OBSOLETE</i>)	bytes	2M	1M	4294967039	N

16.4.5.2. Management Node Configuration Parameters

The following table provides information about parameters used in the [\[NDB_MGMD\]](#) or [\[MGM\]](#) sections of a [config.ini](#) file for configuring MySQL Cluster management nodes. For detailed descriptions and other additional information about each of these parameters, see [Section 16.4.4.4, “Defining the Management Server”](#).

Restart Type Column Values

- **N**: Node Restart
- **IN**: Initial Node Restart
- **S**: System Restart
- **IS**: Initial System Restart

See [Section 16.4.5, “Overview of Cluster Configuration Parameters”](#), for additional explanations of these abbreviations.

Parameter Name	Type / Units	Default Value	Minimum Value	Maximum Value	Restart Type
ArbitrationDelay	milliseconds	0	0	4294967039	N
ArbitrationRank	integer	1	0	2	N
DataDir	string	./ (ndb_mgmd directory)	N/A	N/A	IN
ExecuteOnComputer	integer				
HostName	string	localhost	N/A	N/A	IN
Id	integer	<i>None</i>	1	63	IN

LogDestination	CON SOL E, SYS LOG, or FIL E	FILE (see Section 16.4.4.4, “Defining the Management Server”)	N/A	N/A	N

16.4.5.3. SQL Node and API Node Configuration Parameters

The following table provides information about parameters used in the [\[SQL\]](#) and [\[API\]](#) sections of a [config.ini](#) file for configuring MySQL Cluster SQL nodes and API nodes. For detailed descriptions and other additional information about each of these parameters, see [Section 16.4.4.6, “Defining SQL and Other API Nodes”](#).

Restart Type Column Values

- **N**: Node Restart
- **IN**: Initial Node Restart
- **S**: System Restart
- **IS**: Initial System Restart

See [Section 16.4.5, “Overview of Cluster Configuration Parameters”](#), for additional explanations of these abbreviations.

Parameter Name	Type / Units	Default Value	Minimum Value	Maximum Value	Restart Type
ArbitrationDelay	milli-seconds	0	0	4294967039	N
ArbitrationRank	integer	1	0	2	N
BatchByteSize	bytes	32K	1K	1M	N
BatchSize	integer	64	1	992	N
ExecuteOnComputer	integer				
HostName	string	localhost	N/A	N/A	IN
Id	integer	<i>None</i>	1	63	IN
MaxScanBatchSize	bytes	256K	32K	16M	N

16.4.6. Configuring Parameters for Local Checkpoints

The parameters discussed in [Logging and Checkpointing](#) and in [Data Memory, Index Memory, and String Memory](#) that are used to configure local checkpoints for a MySQL Cluster do not exist in isolation, but rather are very much interdependent on each other. In this section, we illustrate how these parameters — including [DataMemory](#), [IndexMemory](#), [NoOfDiskPagesToDiskAfterRestartTUP](#), [NoOfDiskPagesToDiskAfterRestartACC](#), and [NoOfFragmentLogFiles](#) — relate to one another in a working Cluster.

Important

The parameters `NoOfDiskPagesToDiskAfterRestartTUP` and `NoOfDiskPagesToDiskAfterRestartACC` were deprecated in MySQL 5.1.6. From MySQL 5.1.6 through 5.1.11, disk writes during LCPs took place at the maximum speed possible. Beginning with MySQL 5.1.12, the speed and throughput for LCPs are controlled using the parameters `DiskSyncSize`, `DiskCheckpointSpeed`, and `DiskCheckpointSpeedInRestart`. See [Section 16.4.4.5, “Defining Data Nodes”](#).

In this example, we assume that our application performs the following numbers of types of operations per hour:

- 50000 selects
- 15000 inserts
- 15000 updates
- 15000 deletes

We also make the following assumptions about the data used in the application:

- We are working with a single table having 40 columns.
- Each column can hold up to 32 bytes of data.
- A typical `UPDATE` run by the application affects the values of 5 columns.
- No `NULL` values are inserted by the application.

A good starting point is to determine the amount of time that should elapse between local checkpoints (LCPs). It worth noting that, in the event of a system restart, it takes 40-60 percent of this interval to execute the REDO log — for example, if the time between LCPs is 5 minutes (300 seconds), then it should take 2 to 3 minutes (120 to 180 seconds) for the REDO log to be read.

The maximum amount of data per node can be assumed to be the size of the `DataMemory` parameter. In this example, we assume that this is 2 GB. The `NoOfDiskPagesToDiskAfterRestartTUP` parameter represents the amount of data to be checkpointed per unit time — however, this parameter is actually expressed as the number of 8K memory pages to be checkpointed per 100 milliseconds. 2 GB per 300 seconds is approximately 6.8 MB per second, or 700 KB per 100 milliseconds, which works out to roughly 85 pages per 100 milliseconds.

Similarly, we can calculate `NoOfDiskPagesToDiskAfterRestartACC` in terms of the time for local checkpoints and the amount of memory required for indexes — that is, the `IndexMemory`. Assuming that we allow 512 MB for indexes, this works out to approximately 20 8-KB pages per 100 milliseconds for this parameter.

Next, we need to determine the number of REDO log files required — that is, fragment log files — the corresponding parameter being `NoOfFragmentLogFiles`. We need to make sure that there are sufficient REDO log files for keeping records for at least 3 local checkpoints. In a production setting, there are always uncertainties — for instance, we cannot be sure that disks always operate at top speed or with maximum throughput. For this reason, it is best to err on the side of caution, so we double our requirement and calculate a number of fragment log files which should be enough to keep records covering 6 local checkpoints.

It is also important to remember that the disk also handles writes to the REDO log, so if you find that the amount of data being written to disk as determined by the values of `NoOfDiskPagesToDiskAfterRestartACC` and `NoOfDiskPagesToDiskAfterRestartTUP` is approaching the amount of disk bandwidth available, you may wish to increase the time between local checkpoints.

Given 5 minutes (300 seconds) per local checkpoint, this means that we need to support writing log records at maximum speed for $6 * 300 = 1800$ seconds. The size of a REDO log record is 72 bytes plus 4 bytes per updated column value plus the maximum size of the updated column, and there is one REDO log record for each table record updated in a transaction, on each node where the data reside. Using the numbers of operations set out previously in this section, we derive the following:

- 50000 select operations per hour yields 0 log records (and thus 0 bytes), since `SELECT` statements are not recorded in the REDO log.
- 15000 `DELETE` statements per hour is approximately 5 delete operations per second. (Since we wish to be conservative in our estimate, we round up here and in the following calculations.) No columns are updated by deletes, so these statements consume only 5 operations * 72 bytes per operation = 360 bytes per second.

- 15000 **UPDATE** statements per hour is roughly the same as 5 updates per second. Each update uses 72 bytes, plus 4 bytes per column * 5 columns updated, plus 32 bytes per column * 5 columns — this works out to $72 + 20 + 160 = 252$ bytes per operation, and multiplying this by 5 operation per second yields 1260 bytes per second.
- 15000 **INSERT** statements per hour is equivalent to 5 insert operations per second. Each insert requires REDO log space of 72 bytes, plus 4 bytes per record * 40 columns, plus 32 bytes per column * 40 columns, which is $72 + 160 + 1280 = 1512$ bytes per operation. This times 5 operations per second yields 7560 bytes per second.

So the total number of REDO log bytes being written per second is approximately $0 + 360 + 1260 + 7560 = 9180$ bytes. Multiplied by 1800 seconds, this yields 16524000 bytes required for REDO logging, or approximately 15.75 MB. The unit used for **NoOfFrag-mentLogFiles** represents a set of 4 16-MB log files — that is, 64 MB. Thus, the minimum value (3) for this parameter is sufficient for the scenario envisioned in this example, since 3 times 64 = 192 MB, or about 12 times what is required; the default value of 8 (or 512 MB) is more than ample in this case.

16.5. Upgrading and Downgrading MySQL Cluster

This portion of the MySQL Cluster chapter covers upgrading and downgrading a MySQL Cluster from one MySQL release to another. It discusses different types of Cluster upgrades and downgrades, and provides a Cluster upgrade/downgrade compatibility matrix (see [Section 16.5.2, “Cluster Upgrade and Downgrade Compatibility”](#)). You are expected already to be familiar with installing and configuring a MySQL Cluster prior to attempting an upgrade or downgrade. See [Section 16.4, “MySQL Cluster Configuration”](#).

For information about upgrading or downgrading between MySQL Cluster 5.1 Carrier Grade Edition releases, or between MySQL Cluster 5.1 Carrier Grade Edition releases and mainline MySQL releases, see the changelogs relating to MySQL Cluster 5.1 Carrier Grade Edition.

This section remains in development, and continues to be updated and expanded.

16.5.1. Performing a Rolling Restart of the Cluster

This section discusses how to perform a *rolling restart* of a MySQL Cluster installation, so called because it involves stopping and starting (or restarting) each node in turn, so that the cluster itself remains operational. This is often done as part of a *rolling upgrade* or *rolling downgrade*, where high availability of the cluster is mandatory and no downtime of the cluster as a whole is permissible. Where we refer to upgrades, the information provided here also generally applies to downgrades as well.

There are a number of reasons why a rolling restart might be desirable:

- **Cluster configuration change.** To make a change in the cluster's configuration, such as adding an SQL node to the cluster, or setting a configuration parameter to a new value.
- **Cluster software upgrade/downgrade.** To upgrade the cluster to a newer version of the MySQL Cluster software (or to downgrade it to an older version). This is usually referred to as a “rolling upgrade” (or “rolling downgrade”, when reverting to an older version of MySQL Cluster).
- **Change on node host.** To make changes in the hardware or operating system on which one or more cluster nodes are running
- **Cluster reset.** To reset the cluster because it has reached an undesirable state
- **Freeing of resources.** To allow memory allocated to a table by successive **INSERT** and **DELETE** operations to be freed for re-use by other Cluster tables

The process for performing a rolling restart may be generalised as follows:

1. Stop all cluster management nodes (**ndb_mgmd** processes), reconfigure them, then restart them
2. Stop, reconfigure, then restart each cluster data node (**ndbd** process) in turn
3. Stop, reconfigure, then restart each cluster SQL node (**mysqld** process) in turn

The specifics for implementing a particular rolling upgrade depend upon the actual changes being made. A more detailed view of the process is presented here:

RESTART TYPE:					
Cluster Configuration Change		Cluster Software Upgrade or Downgrade	Change on Node Host	Cluster Reset	
A. Management node (ndb_mgmd) processes...					
1. Stop all ndb_mgmd processes 2. Make changes in global configuration file(s) 3. Start all ndb_mgmd processes		1. Stop all ndb_mgmd processes 2. Replace each ndb_mgmd binary with new version 3. Start ndb_mgmd processes	1. Stop all ndb_mgmd processes 2. Make desired changes in hardware, operating system, or both 3. Start all ndb_mgmd processes	(OR)	
				1. Stop all ndb_mgmd processes 2. Start all ndb_mgmd processes	Restart all ndb_mgmd processes (optional)
B. For each data node (ndbd) process...					
(OR)		1. Stop ndbd 2. Replace ndbd binary with new version 3. Start ndbd	1. Stop ndbd 2. Make desired changes in hardware, operating system, or both 3. Start ndbd	(OR)	
1. Stop ndbd 2. Start ndbd	Restart ndbd			1. Stop ndbd 2. Start ndbd	Restart ndbd
C. For each SQL node (mysqld) process...					
(OR)		1. Stop mysqld 2. Replace mysqld binary with new version 3. Start mysqld	1. Stop mysqld 2. Make desired changes in hardware, operating system, or both 3. Start mysqld	(OR)	
1. Stop mysqld 2. Start mysqld	Restart mysqld			1. Stop mysqld 2. Start mysqld	Restart mysqld

In the previous diagram, **Stop** and **Start** steps indicate that the process must be stopped completely using a shell command (such as `kill` on most Unix systems) or the management client `STOP` command, then started again from a system shell by invoking the `ndbd` or `ndb_mgmd` executable as appropriate. **Restart** indicates the process may be restarted using the `ndb_mgm` management client `RE-START` command.

Important

When performing an upgrade or downgrade of the cluster software, you *must* upgrade or downgrade the management nodes *first*, then the data nodes, and finally the SQL nodes. Doing so in any other order may leave the cluster in an unus-

■ able state.

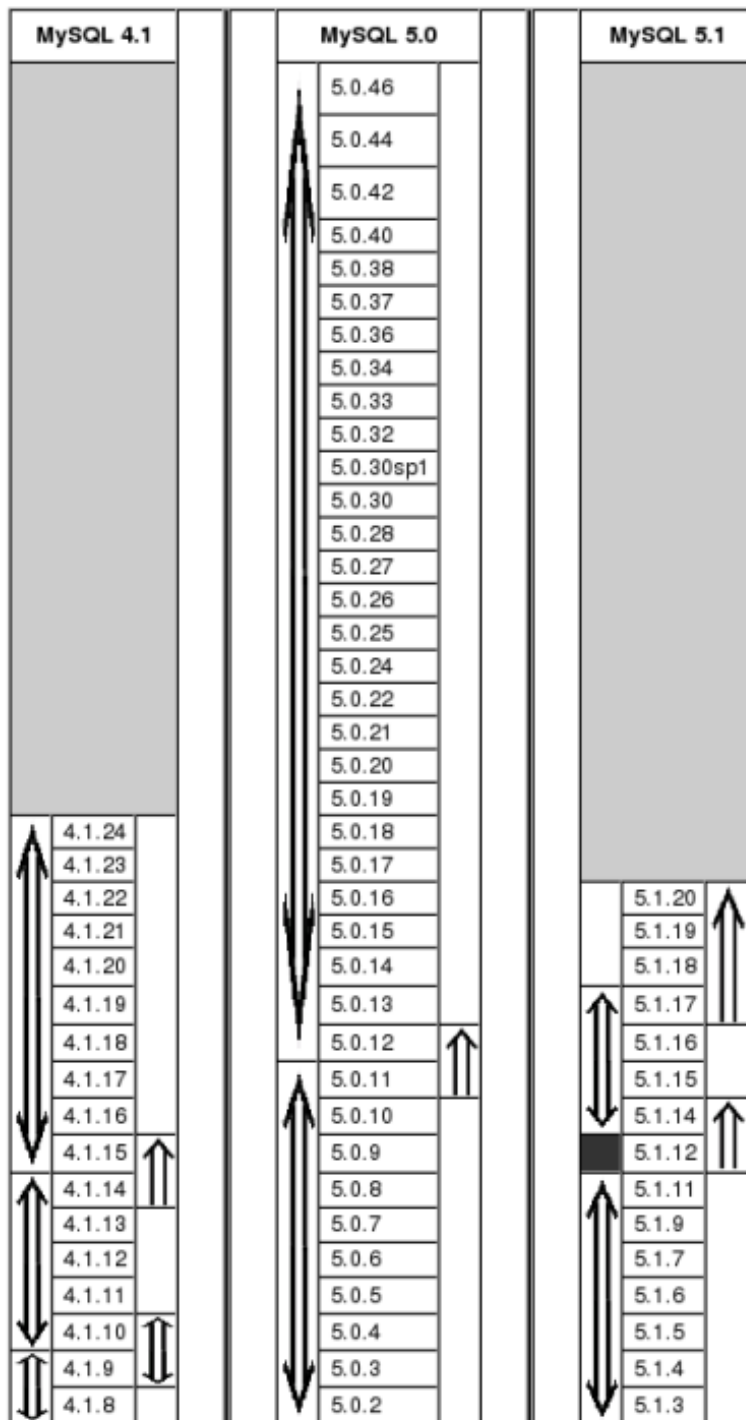
16.5.2. Cluster Upgrade and Downgrade Compatibility

This section provides information regarding Cluster software and table file compatibility between differing versions of the MySQL Server for purposes of performing upgrades and downgrades.

Important

Only compatibility between MySQL versions with regard to [NDB Cluster](#) is taken into account in this section, and there are likely other issues to be considered. *As with any other MySQL software upgrade or downgrade, you are strongly encouraged to review the relevant portions of the MySQL Manual for the MySQL versions from which and to which you intend to migrate, before attempting an upgrade or downgrade of the MySQL Cluster software.* See [Section 2.11](#), “Upgrading MySQL”.

The following table shows Cluster upgrade and downgrade compatibility between different versions of the MySQL Server.

**KEY:**

↑↓ Upgrades and downgrades supported

↑ Upgrade only

Notes:

- **4.1 Series:**

You cannot upgrade directly from 4.1.8 to 4.1.10 (or newer); you must first upgrade from 4.1.8 to 4.1.9, then upgrade to 4.1.10. Similarly, you cannot downgrade directly from 4.1.10 (or newer) to 4.1.8; you must first downgrade from 4.1.10 to 4.1.9, then downgrade from 4.1.9 to 4.1.8.

If you wish to upgrade a MySQL Cluster to 4.1.15, you must upgrade to 4.1.14 first, and you must upgrade to 4.1.15 before upgrading to 4.1.16 or newer.

Cluster downgrades from 4.1.15 to 4.1.14 (or earlier versions) are not supported.

Cluster upgrades from MySQL Server versions previous to 4.1.8 are not supported; when upgrading from these, you must dump all **NDB** tables, install the new version of the software, and then reload the tables from the dump.

- **5.0 Series:**

MySQL 5.0.2 was the first public release in this series.

Cluster downgrades from MySQL 5.0 to MySQL 4.1 are not supported.

Cluster downgrades from 5.0.12 to 5.0.11 (or earlier) are not supported.

You cannot restore with **ndb_restore** to a MySQL 5.0 Cluster using a backup made from a Cluster running MySQL 5.1. You must use **mysqldump** in such cases.

There was no public release for MySQL 5.0.23.

- **5.1 Series:**

MySQL 5.1.3 was the first public release in this series.

You cannot downgrade a MySQL 5.1.6 or later Cluster using Disk Data tables to MySQL 5.1.5 or earlier unless you convert all such tables to in-memory Cluster tables first.

MySQL 5.1.8, MySQL 5.1.10, and MySQL 5.1.13 were not released.

Online cluster upgrades and downgrades between MySQL 5.1.11 (or an earlier version) and 5.1.12 (or a later version) are not possible due to major changes in the cluster filesystem. In such cases, you must perform a backup or dump, upgrade (or downgrade) the software, start each data node with **--initial**, and then restore from the backup or dump. You can use **NDB** backup/restore or **mysqldump** for this purpose.

- Online downgrades from MySQL 5.1.14 or later to versions previous to 5.1.14 are not supported due to incompatible changes in the cluster system tables.
- Online upgrades from MySQL 5.1.17 and earlier to 5.1.18 and later are not supported for clusters using replication due to incompatible changes in the **mysql.ndb_apply_status** table. However, it should not be necessary to shut down the cluster entirely, if you follow this modified rolling restart procedure:

1. Stop the management server, update the **ndb_mgmd** binary, then start it again. For multiple management servers, repeat this step for each management server in turn.
2. For each data node in turn: Stop the data node, replace the **ndbd** binary with the new version, then restart the data node. It is not necessary to use **--initial** when restarting any of the data nodes.
3. Stop *all* SQL nodes. Replace the **mysqld** binary with the new version for all SQL nodes, then restart them. It is not necessary to start them one at a time, but they must all be shut down at the same time before starting any of them again using the 5.1.18 (or later) **mysqld**. Otherwise — due to the fact that **mysql.ndb_apply_status** uses the **NDB** storage engine and is thus shared between all SQL nodes — there may be conflicts between MySQL servers using the old and new versions of the table. You can find more information about the changes to **ndb_apply_status** in [Section 16.10.4, “Cluster Replication Schema and Tables”](#).

- The internal specifications for columns in **NDB** tables changed in MySQL 5.1.18 to allow compatibility with future MySQL Cluster releases that are expected to implement online adding and dropping of columns. This change is not backwards compatible with earlier

er MySQL versions.

In order to make tables created in MySQL 5.1.17 and earlier compatible with online adding and dropping of columns when this feature becomes available, it is necessary for MySQL 5.1.18 and later to convert the tables to the new format by following this procedure:

1. Upgrade the MySQL Cluster software on all data, management, and SQL nodes
2. Back up all **NDB** tables
3. Shut down the cluster (all data, management, and SQL nodes)
4. Restart the cluster, starting all data nodes with the `--initial` option (to clear and rebuild the data node filesystems)
5. Restore the tables from backup

This is not necessary for **NDB** tables created in MySQL 5.1.18 and later; such tables will automatically be compatible with online adding and dropping of columns when this feature is introduced.

In order to minimise possible later difficulties, it is strongly advised that the procedure outlined above be followed as soon as possible after upgrading from MySQL 5.1.17 or earlier to MySQL 5.1.18 or later.

For users of MySQL Cluster 5.1 Carrier Grade Edition the relevant versions are as follows:

- **ndb-6.1.x series.** The new table format is implemented beginning with MySQL 5.1.15-ndb-6.1.7.
- **ndb-6.2.x series.** The new table format is implemented beginning with MySQL 5.1.16-ndb-6.2.1.

16.6. Process Management in MySQL Cluster

Understanding how to manage MySQL Cluster requires a knowledge of four essential processes. In the next few sections of this chapter, we cover the roles played by these processes in a cluster, how to use them, and what startup options are available for each of them:

- [Section 16.6.1, “MySQL Server Process Usage for MySQL Cluster”](#)
- [Section 16.6.2, “ndbd — The Storage Engine Node Process”](#)
- [Section 16.6.3, “ndb_mgmd — The Management Server Process”](#)
- [Section 16.6.4, “ndb_mgm — The Management Client Process”](#)

16.6.1. MySQL Server Process Usage for MySQL Cluster

`mysqld` is the traditional MySQL server process. To be used with MySQL Cluster, `mysqld` needs to be built with support for the **NDB Cluster** storage engine, as it is in the precompiled binaries available from <http://dev.mysql.com/downloads/>. If you build MySQL from source, you must invoke `configure` with the `--with-ndbcluster` option to enable **NDB Cluster** storage engine support.

If the `mysqld` binary has been built with Cluster support, the **NDB Cluster** storage engine is still disabled by default. You can use either of two possible options to enable this engine:

- Use `--ndbcluster` as a startup option on the command line when starting `mysqld`.
- Insert a line containing `ndbcluster` in the `[mysqld]` section of your `my.cnf` file.

An easy way to verify that your server is running with the **NDB Cluster** storage engine enabled is to issue the `SHOW ENGINES` statement in the MySQL Monitor (`mysql`). You should see the value **YES** as the **Support** value in the row for **NDBCLUSTER**. If you see **NO** in this row or if there is no such row displayed in the output, you are not running an **NDB-enabled** version of MySQL. If you see **DISABLED** in this row, you need to enable it in either one of the two ways just described.

To read cluster configuration data, the MySQL server requires at a minimum three pieces of information:

- The MySQL server's own cluster node ID
- The hostname or IP address for the management server (MGM node)
- The number of the TCP/IP port on which it can connect to the management server

Node IDs can be allocated dynamically, so it is not strictly necessary to specify them explicitly.

The `mysqld` parameter `ndb-connectstring` is used to specify the connectstring either on the command line when starting `mysqld` or in `my.cnf`. The connectstring contains the hostname or IP address where the management server can be found, as well as the TCP/IP port it uses.

In the following example, `ndb_mgmd.mysql.com` is the host where the management server resides, and the management server listens for cluster messages on port 1186:

```
shell> mysqld --ndbcluster --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

See [Section 16.4.4.2, “The Cluster Connectstring”](#), for more information on connectstrings.

Given this information, the MySQL server will be a full participant in the cluster. (We often refer to a `mysqld` process running in this manner as an SQL node.) It will be fully aware of all cluster data nodes as well as their status, and will establish connections to all data nodes. In this case, it is able to use any data node as a transaction coordinator and to read and update node data.

You can see in the `mysql` client whether a MySQL server is connected to the cluster using `SHOW PROCESSLIST`. If the MySQL server is connected to the cluster, and you have the `PROCESS` privilege, then the first row of the output is as shown here:

```
mysql> SHOW PROCESSLIST \G
***** 1. row *****
  Id: 1
  User: system user
  Host:
  db:
  Command: Daemon
  Time: 1
  State: Waiting for event from ndbcluster
  Info: NULL
```

Important

To participate in a MySQL Cluster, the `mysqld` process must be started with *both* the options `--ndbcluster` and `--ndb-connectstring` (or their equivalents in `my.cnf`). If `mysqld` is started with only the `--ndbcluster` option, or if it is unable to contact the cluster, it is not possible to work with NDB tables, *nor is it possible to create any new tables regardless of storage engine*. The latter restriction is a safety measure intended to prevent the creation of tables having the same names as NDB tables while the SQL node is not connected to the cluster. If you wish to create tables using a different storage engine while the `mysqld` process is not participating in a MySQL Cluster, you must restart the server *without* the `--ndbcluster` option.

16.6.2. ndbd — The Storage Engine Node Process

`ndbd` is the process that is used to handle all the data in tables using the NDB Cluster storage engine. This is the process that empowers a data node to accomplish distributed transaction handling, node recovery, checkpointing to disk, online backup, and related tasks.

In a MySQL Cluster, a set of `ndbd` processes cooperate in handling data. These processes can execute on the same computer (host) or on different computers. The correspondences between data nodes and Cluster hosts is completely configurable.

`ndbd` generates a set of log files which are placed in the directory specified by `DataDir` in the `config.ini` configuration file.

These log files are listed below. `node_id` is the node's unique identifier. Note that `node_id` represents the node's unique identifier. For example, `ndb_2_error.log` is the error log generated by the data node whose node ID is 2.

- `ndb_node_id_error.log` is a file containing records of all crashes which the referenced `ndbd` process has encountered. Each

record in this file contains a brief error string and a reference to a trace file for this crash. A typical entry in this file might appear as shown here:

```
Date/Time: Saturday 30 July 2004 - 00:20:01
Type of error: error
Message: Internal program error (failed ndbrequire)
Fault ID: 2341
Problem data: DbtupFixAlloc.cpp
Object of reference: DBTUP (Line: 173)
ProgramName: NDB Kernel
ProcessID: 14909
TraceFile: ndb_2_trace.log.2
***EOM***
```

Listings of possible `ndbd` exit codes and messages generated when a data node process shuts down prematurely can be found in [ndbd Error Messages](#).

Important

The last entry in the error log file is not necessarily the newest one (nor is it likely to be). Entries in the error log are not listed in chronological order; rather, they correspond to the order of the trace files as determined in the [ndb_node_id_trace.log.next](#) file (see below). Error log entries are thus overwritten in a cyclical and not sequential fashion.

- [ndb_node_id_trace.log.trace_id](#) is a trace file describing exactly what happened just before the error occurred. This information is useful for analysis by the MySQL Cluster development team.
- It is possible to configure the number of these trace files that will be created before old files are overwritten. [trace_id](#) is a number which is incremented for each successive trace file.
- [ndb_node_id_trace.log.next](#) is the file that keeps track of the next trace file number to be assigned.
- [ndb_node_id_out.log](#) is a file containing any data output by the `ndbd` process. This file is created only if `ndbd` is started as a daemon, which is the default behavior.
- [ndb_node_id.pid](#) is a file containing the process ID of the `ndbd` process when started as a daemon. It also functions as a lock file to avoid the starting of nodes with the same identifier.
- [ndb_node_id_signal.log](#) is a file used only in debug versions of `ndbd`, where it is possible to trace all incoming, outgoing, and internal messages with their data in the `ndbd` process.

It is recommended not to use a directory mounted through NFS because in some environments this can cause problems whereby the lock on the `.pid` file remains in effect even after the process has terminated.

To start `ndbd`, it may also be necessary to specify the hostname of the management server and the port on which it is listening. Optionally, one may also specify the node ID that the process is to use.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

See [Section 16.4.4.2, “The Cluster Connectstring”](#), for additional information about this issue. [Section 16.6.5, “Command Options for MySQL Cluster Processes”](#), describes other options for `ndbd`.

When `ndbd` starts, it actually initiates two processes. The first of these is called the “angel process”; its only job is to discover when the execution process has been completed, and then to restart the `ndbd` process if it is configured to do so. Thus, if you attempt to kill `ndbd` via the Unix `kill` command, it is necessary to kill both processes, beginning with the angel process. The preferred method of terminating an `ndbd` process is to use the management client and stop the process from there.

The execution process uses one thread for reading, writing, and scanning data, as well as all other activities. This thread is implemented asynchronously so that it can easily handle thousands of concurrent activities. In addition, a watch-dog thread supervises the execution thread to make sure that it does not hang in an endless loop. A pool of threads handles file I/O, with each thread able to handle one open file. Threads can also be used for transporter connections by the transporters in the `ndbd` process. In a multi-processor system performing a large number of operations (including updates), the `ndbd` process can consume up to 2 CPUs if permitted to do so.

For a machine with many CPUs it is possible to use several `ndbd` processes which belong to different node groups; however, such a configuration is still considered experimental and is not supported for MySQL 5.1 in a production setting. See [Section 16.13, “Known](#)

Limitations of MySQL Cluster”.

16.6.3. **ndb_mgmd** — The Management Server Process

The management server is the process that reads the cluster configuration file and distributes this information to all nodes in the cluster that request it. It also maintains a log of cluster activities. Management clients can connect to the management server and check the cluster's status.

It is not strictly necessary to specify a connectstring when starting the management server. However, if you are using more than one management server, a connectstring should be provided and each node in the cluster should specify its node ID explicitly.

See [Section 16.4.4.2, “The Cluster Connectstring”](#), for information about using connectstrings. [Section 16.6.5, “Command Options for MySQL Cluster Processes”](#), describes other options for **ndb_mgmd**.

The following files are created or used by **ndb_mgmd** in its starting directory, and are placed in the **DataDir** as specified in the **config.ini** configuration file. In the list that follows, *node_id* is the unique node identifier.

- **config.ini** is the configuration file for the cluster as a whole. This file is created by the user and read by the management server. [Section 16.4, “MySQL Cluster Configuration”](#), discusses how to set up this file.

- **ndb_node_id_cluster.log** is the cluster events log file. Examples of such events include checkpoint startup and completion, node startup events, node failures, and levels of memory usage. A complete listing of cluster events with descriptions may be found in [Section 16.7, “Management of MySQL Cluster”](#).

When the size of the cluster log reaches one million bytes, the file is renamed to **ndb_node_id_cluster.log.seq_id**, where *seq_id* is the sequence number of the cluster log file. (For example: If files with the sequence numbers 1, 2, and 3 already exist, the next log file is named using the number 4.)

- **ndb_node_id_out.log** is the file used for **stdout** and **stderr** when running the management server as a daemon.
- **ndb_node_id.pid** is the process ID file used when running the management server as a daemon.

16.6.4. **ndb_mgm** — The Management Client Process

The **ndb_mgm** management client process is actually not needed to run the cluster. Its value lies in providing a set of commands for checking the cluster's status, starting backups, and performing other administrative functions. The management client accesses the management server using a C API. Advanced users can also employ this API for programming dedicated management processes to perform tasks similar to those performed by **ndb_mgm**.

To start the management client, it is necessary to supply the hostname and port number of the management server:

```
shell> ndb_mgm [host_name [port_num]]
```

For example:

```
shell> ndb_mgm ndb_mgmd.mysql.com 1186
```

The default hostname and port number are **localhost** and 1186, respectively.

Additional information about using **ndb_mgm** can be found in [Section 16.6.5.4, “Command Options for ndb_mgm”](#), and [Section 16.7.2, “Commands in the MySQL Cluster Management Client”](#).

16.6.5. Command Options for MySQL Cluster Processes

All MySQL Cluster executables (except for **mysqld**) take the options described in this section. Users of earlier MySQL Cluster versions should note that some of these options have been changed to make them consistent with one another as well as with **mysqld**. You can use the **--help** option with any MySQL Cluster executable to view a list of the options which it supports.

The following options are common to all MySQL Cluster executables:

-

`--help --usage, -?`

Prints a short list with descriptions of the available command options.

- `--connect-string=connect_string, -c connect_string`

`connect_string` sets the connectstring to the management server as a command option.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

- `--debug[=options]`

This option can be used only for versions compiled with debugging enabled. It is used to enable output from debug calls in the same manner as for the `mysqld` process.

- `--execute=command, -e command`

Can be used to send a command to a Cluster executable from the system shell. For example, either of the following:

```
shell> ndb_mgm -e "SHOW"
```

or

```
shell> ndb_mgm --execute="SHOW"
```

is equivalent to

```
ndb_mgm> SHOW
```

This is analogous to how the `--execute` or `-e` option works with the `mysql` command-line client. See [Section 4.3.1, “Using Options on the Command Line”](#).

- `--version, -V`

Prints the MySQL Cluster version number of the executable. The version number is relevant because not all versions can be used together, and the MySQL Cluster startup process verifies that the versions of the binaries being used can co-exist in the same cluster. This is also important when performing an online (rolling) software upgrade or downgrade of MySQL Cluster. (See [Section 16.5.1, “Performing a Rolling Restart of the Cluster”](#)).

The next few sections describe options specific to individual [NDB](#) programs.

16.6.5.1. MySQL Cluster-Related Command Options for `mysqld`

- `--ndb-connectstring=connect_string`

When using the [NDB Cluster](#) storage engine, this option specifies the management server that distributes cluster configuration data.

- `--ndbcluster`

The [NDB Cluster](#) storage engine is necessary for using MySQL Cluster. If a `mysqld` binary includes support for the [NDB Cluster](#) storage engine, the engine is disabled by default. Use the `--ndbcluster` option to enable it. Use `--skip-ndbcluster` to explicitly disable the engine.

16.6.5.2. Command Options for `ndbd`

For options common to all `NDB` programs, see [Section 16.6.5, “Command Options for MySQL Cluster Processes”](#).

- `--bind-address`

Causes `ndbd` to bind to a specific network interface. This option has no default value.

This option was added in MySQL 5.1.12.

- `--daemon, -d`

Instructs `ndbd` to execute as a daemon process. This is the default behavior. `--nodaemon` can be used to prevent the process from running as a daemon.

- `--initial`

Instructs `ndbd` to perform an initial start. An initial start erases any files created for recovery purposes by earlier instances of `ndbd`. It also re-creates recovery log files. Note that on some operating systems this process can take a substantial amount of time.

An `--initial` start is to be used *only* when starting the `ndbd` process under very special circumstances; this is because this option causes all files to be removed from the Cluster filesystem and all redo log files to be re-created. These circumstances are listed here:

- When performing a software upgrade which has changed the contents of any files.
- When restarting the node with a new version of `ndbd`.
- As a measure of last resort when for some reason the node restart or system restart repeatedly fails. In this case, be aware that this node can no longer be used to restore data due to the destruction of the data files.

Important

This option does *not* affect either of the following:

- Backup files that have already been created by the affected node
- Cluster Disk Data files (see [Section 16.11, “MySQL Cluster Disk Data Tables”](#)).

It is permissible to use this option when starting the cluster for the very first time (that is, before any data node files have been created); however, it is *not* necessary to do so.

- `--initial-start`

This option is used when performing a partial initial start of the cluster. Each node should be started with this option, as well as `--nowait-nodes`.

For example, suppose you have a 4-node cluster whose data nodes have the IDs 2, 3, 4, and 5, and you wish to perform a partial initial start using only nodes 2, 4, and 5 — that is, omitting node 3:

```
ndbd --ndbd-nodeid=2 --nowait-nodes=3 --initial-start
ndbd --ndbd-nodeid=4 --nowait-nodes=3 --initial-start
ndbd --ndbd-nodeid=5 --nowait-nodes=3 --initial-start
```

This option was added in MySQL 5.1.11.

Important

Prior to MySQL 5.1.19, it was not possible to perform DDL operations involving Disk Data tables on a partially started cluster. (See [Bug#24631](#).)

- `--nowait-nodes=node_id_1[, node_id_2[, ...]]`

This option takes a list of data nodes which for which the cluster will not wait for before starting.

This can be used to start the cluster in a partitioned state. For example, to start the cluster with only half of the data nodes (nodes 2, 3, 4, and 5) running in a 4-node cluster, you can start each `ndbd` process with `--nowait-nodes=3,5`. In this case, the cluster starts as soon as nodes 2 and 4 connect, and does *not* wait `StartPartitionedTimeout` milliseconds for nodes 3 and 5 to connect as it would otherwise.

If you wanted to start up the same cluster as in the previous example without one `ndbd` — say, for example, that the host machine for node 3 has suffered a hardware failure — then start nodes 2, 4, and 5 with `--nowait-nodes=3`. Then the cluster will start as soon as nodes 2, 4, and 5 connect and will not wait for node 3 to start.

This option was added in MySQL 5.1.9.

- `--nodaemon`
Instructs `ndbd` not to start as a daemon process. This is useful when `ndbd` is being debugged and you want output to be redirected to the screen.
- `--nostart, -n`
Instructs `ndbd` not to start automatically. When this option is used, `ndbd` connects to the management server, obtains configuration data from it, and initializes communication objects. However, it does not actually start the execution engine until specifically requested to do so by the management server. This can be accomplished by issuing the proper `START` command in the management client (see Section 16.7.2, “Commands in the MySQL Cluster Management Client”).

16.6.5.3. Command Options for `ndb_mgmd`

For options common to NDB programs, see Section 16.6.5, “Command Options for MySQL Cluster Processes”.

- **MySQL Cluster 5.1 Carrier Grade Edition**
The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

`--bind-address=host[:port]`
When specified, this option limits management server connections by management clients to clients at the specified hostname or IP address (and possibly port, if this is also specified). In such cases, a management client attempting to connect to the management server from any other address fails with the error `UNABLE TO SETUP PORT: HOST:PORT!`
If the `port` is not specified, the management client attempts to use port 1186.
This option was added in MySQL 5.1.19-ndb-6.2.5 and MySQL 5.1.22-ndb-6.3.2. It is not available in MySQL Cluster 5.1 Carrier Grade Edition ndb-6.1.x or mainline MySQL 5.1 releases.

The following information applies to all MySQL Cluster users.

`--config-file=filename, -f filename`
Instructs the management server as to which file it should use for its configuration file. This option must be specified. The filename defaults to `config.ini`.

Note
This option also can be given as `-c file_name`, but this shortcut is obsolete and should *not* be used in new installations.
- `--daemon, -d`

Instructs `ndb_mgmd` to start as a daemon process. This is the default behavior.

- `--nodaemon`

Instructs `ndb_mgmd` not to start as a daemon process.

16.6.5.4. Command Options for `ndb_mgm`

For options common to NDB programs, see [Section 16.6.5, “Command Options for MySQL Cluster Processes”](#).

- `--try-reconnect=number`

If the connection to the management server is broken, the node tries to reconnect to it every 5 seconds until it succeeds. By using this option, it is possible to limit the number of attempts to *number* before giving up and reporting an error instead.

16.7. Management of MySQL Cluster

Managing a MySQL Cluster involves a number of tasks, the first of which is to configure and start MySQL Cluster. This is covered in [Section 16.4, “MySQL Cluster Configuration”](#), and [Section 16.6, “Process Management in MySQL Cluster”](#).

The following sections cover the management of a running MySQL Cluster.

There are essentially two methods of actively managing a running MySQL Cluster. The first of these is through the use of commands entered into the management client whereby cluster status can be checked, log levels changed, backups started and stopped, and nodes stopped and started. The second method involves studying the contents of the cluster log `ndb_node_id_cluster.log`; this is usually found in the management server's `DataDir` directory, but this location can be overridden using the `LogDestination` option — see [Section 16.4.4.4, “Defining the Management Server”](#), for details. (Recall that *node_id* represents the unique identifier of the node whose activity is being logged.) The cluster log contains event reports generated by `ndbd`. It is also possible to send cluster log entries to a Unix system log.

In addition, some aspects of the cluster's operation can be monitored from an SQL node using the `SHOW ENGINE NDB STATUS` statement. See [Section 12.5.4.13, “SHOW ENGINE Syntax”](#), for more information.

16.7.1. Summary of MySQL Cluster Start Phases

This section provides a simplified outline of the steps involved when MySQL Cluster data nodes are started. More complete information can be found in [MySQL Cluster Start Phases](#).

These phases are the same as those reported in the output from the `node_id STATUS` command in the management client. (See [Section 16.7.2, “Commands in the MySQL Cluster Management Client”](#), for more information about this command.)

Start types. There are several different startup types and modes, as shown here:

- **Initial Start.** The cluster starts with a clean filesystem on all data nodes. This occurs either when the cluster started for the very first time, or when all data nodes are restarted using the `--initial` option.

Note

Disk Data files are not removed when restarting a node using `--initial`.

- **System Restart.** The cluster starts and reads data stored in the data nodes. This occurs when the cluster has been shut down after having been in use, when it is desired for the cluster to resume operations from the point where it left off.
- **Node Restart.** This is the online restart of a cluster node while the cluster itself is running.
- **Initial Node Restart.** This is the same as a node restart, except that the node is reinitialized and started with a clean filesystem.

Setup and initialization (Phase -1). Prior to startup, each data node (`ndbd` process) must be initialized. Initialization consists of the following steps:

1. Obtain a node ID
2. Fetch configuration data
3. Allocate ports to be used for inter-node communications
4. Allocate memory according to settings obtained from the configuration file

When a data node or SQL node first connects to the management node, it reserves a cluster node ID. To make sure that no other node allocates the same node ID, this ID is retained until the node has managed to connect to the cluster and at least one `ndbd` reports that this node is connected. This retention of the node ID is guarded by the connection between the node in question and `ndb_mgmd`.

Normally, in the event of a problem with the node, the node disconnects from the management server, the socket used for the connection is closed, and the reserved node ID is freed. However, if a node is disconnected abruptly — for example, due to a hardware failure in one of the cluster hosts, or because of network issues — the normal closing of the socket by the operating system may not take place. In this case, the node ID continues to be reserved and not released until a TCP timeout occurs 10 or so minutes later.

To take care of this problem, you can use `PURGE STALE SESSIONS`. Running this statement forces all reserved node IDs to be checked; any that are not being used by nodes actually connected to the cluster are then freed.

Beginning with MySQL 5.1.11, timeout handling of node ID assignments is implemented. This performs the ID usage checks automatically after approximately 20 seconds, so that `PURGE STALE SESSIONS` should no longer be necessary in a normal Cluster start.

After each data node has been initialized, the cluster startup process can proceed. The stages which the cluster goes through during this process are listed here:

- **Phase 0.** The `NDBFS` and `NDBCNTR` blocks start (see [NDB Kernel Blocks](#)). The cluster filesystem is cleared, if the cluster was started with the `--initial` option.
- **Phase 1.** In this stage, all remaining `NDB` kernel blocks are started. Cluster connections are set up, inter-block communications are established, and Cluster heartbeats are started. In the case of a node restart, API node connections are also checked.

Note

When one or more nodes hang in Phase 1 while the remaining node or nodes hang in Phase 2, this often indicates network problems. One possible cause of such issues is one or more cluster hosts having multiple network interfaces. Another common source of problems causing this condition is the blocking of TCP/IP ports needed for communications between cluster nodes. In the latter case, this is often due to a misconfigured firewall.

- **Phase 2.** The `NDBCNTR` kernel block checks the states of all existing nodes. The master node is chosen, and the cluster schema file is initialized.
- **Phase 3.** The `DBLQH` and `DBTC` kernel blocks set up communications between them. The startup type is determined; if this is a restart, the `DBDIH` block obtains permission to perform the restart.
- **Phase 4.** For an initial start or initial node restart, the redo log files are created. The number of these files is equal to `NoOfFragmentLogFiles`.

For a system restart:

- Read schema or schemas.
- Read data from the local checkpoint.
- Apply all redo information until the latest restorable global checkpoint has been reached.

For a node restart, find the tail of the redo log.

- **Phase 5.** Most of the database-related portion of a data node start is performed during this phase. For an initial start or system restart, a local checkpoint is executed, followed by a global checkpoint. Periodic checks of memory usage begin during this phase, and any required node takeovers are performed.
- **Phase 6.** In this phase, node groups are defined and set up.

- **Phase 7.** The arbitrator node is selected and begins to function. The next backup ID is set, as is the backup disk write speed. Nodes reaching this start phase are marked as **Started**. It is now possible for API nodes (including SQL nodes) to connect to the cluster.
- **Phase 8.** If this is a system restart, all indexes are rebuilt (by **DBDIIH**).
- **Phase 9.** The node internal startup variables are reset.
- **Phase 100 (OBSOLETE).** Formerly, it was at this point during a node restart or initial node restart that API nodes could connect to the node and begin to receive events. Currently, this phase is empty.
- **Phase 101.** At this point in a node restart or initial node restart, event delivery is handed over to the node joining the cluster. The newly-joined node takes over responsibility for delivering its primary data to subscribers. This phase is also referred to as **SUMA handover phase**.

After this process is completed for an initial start or system restart, transaction handling is enabled. For a node restart or initial node restart, completion of the startup process means that the node may now act as a transaction coordinator.

16.7.2. Commands in the MySQL Cluster Management Client

In addition to the central configuration file, a cluster may also be controlled through a command-line interface available through the management client **ndb_mgm**. This is the primary administrative interface to a running cluster.

Commands for the event logs are given in [Section 16.7.3, “Event Reports Generated in MySQL Cluster”](#); commands for creating backups and restoring from backup are provided in [Section 16.8, “On-line Backup of MySQL Cluster”](#).

The management client has the following basic commands. In the listing that follows, *node_id* denotes either a database node ID or the keyword **ALL**, which indicates that the command should be applied to all of the cluster's data nodes.

- **HELP**
Displays information on all available commands.

- **SHOW**
Displays information on the cluster's status.

Note

In a cluster where multiple management nodes are in use, this command displays information only for data nodes that are actually connected to the current management server.

- **node_id START**
Brings online the data node identified by *node_id* (or all data nodes).
ALL START works on all data nodes only, and does not affect management nodes.

Important

To use this command to bring a data node online, the data node must have been started using **ndbd --nostart** or **ndbd -n**.

- **node_id STOP**
Stops the data or management node identified by *node_id*. Note that **ALL STOP** works to stop all data nodes only, and does not affect management nodes.
A node affected by this command disconnects from the cluster, and its associated **ndbd** or **ndb_mgmd** process terminates.

- `node_id RESTART [-n] [-i] [-a]`

Restarts the data node identified by `node_id` (or all data nodes).

Using the `-i` option with `RESTART` causes the data node to perform an initial restart; that is, the node's filesystem is deleted and re-created. The effect is the same as that obtained from stopping the data node process and then starting it again using `ndbd -initial` from the system shell. Note that backup files and Disk Data files are not removed when this option is used.

Using the `-n` option causes the data node process to be restarted, but the data node is not actually brought online until the appropriate `START` command is issued. The effect of this option is the same as that obtained from stopping the data node and then starting it again using `ndbd --nostart` or `ndbd -n` from the system shell.

Using the `-a` causes all current transactions relying on this node to be aborted. No GCP check is done when the node rejoins the cluster.

- `node_id STATUS`

Displays status information for the data node identified by `node_id` (or for all data nodes).

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, "MySQL Cluster 5.1 Carrier Grade Edition".

`node_id REPORT report-type`

Displays a report of type `report-type` for the data node identified by `node_id`, or for all data nodes using `ALL`.

Currently, the only accepted value for `report-type` is `BackupStatus`, which provides a status report on a cluster backup in progress.

The `REPORT` command was introduced in MySQL 5.1.8-ndb-6.2.3.

The following information applies to all MySQL Cluster users.

- `ENTER SINGLE USER MODE node_id`

Enters single user mode, whereby only the MySQL server identified by the node ID `node_id` is allowed to access the database.

Important

It is not possible in MySQL 5.1 for data nodes to join the cluster while it is running in single user mode. (See [Bug#20395](#) for more information.)

- `EXIT SINGLE USER MODE`

Exits single user mode, allowing all SQL nodes (that is, all running `mysqld` processes) to access the database.

- `QUIT, EXIT`

Terminates the management client.

This command does not affect any nodes connected to the cluster.

- `SHUTDOWN`

Shuts down all cluster data nodes and management nodes. To exit the management client after this has been done, use `EXIT` or `QUIT`.

This command does *not* shut down any SQL nodes or API nodes that are connected to the cluster.

16.7.3. Event Reports Generated in MySQL Cluster

In this section, we discuss the types of event logs provided by MySQL Cluster, and the types of events that are logged.

MySQL Cluster provides two types of event log:

- The *cluster log*, which includes events generated by all cluster nodes. The cluster log is the log recommended for most uses because it provides logging information for an entire cluster in a single location.

By default, the cluster log is saved to a file named `ndb_node_id_cluster.log`, (where *node_id* is the node ID of the management server) in the same directory where the `ndb_mgm` binary resides.

Cluster logging information can also be sent to `stdout` or a `syslog` facility in addition to or instead of being saved to a file, as determined by the values set for the `DataDir` and `LogDestination` configuration parameters. See [Section 16.4.4.4, “Defining the Management Server”](#), for more information about these parameters.

- *Node logs* are local to each node.

Output generated by node event logging is written to the file `ndb_node_id_out.log` (where *node_id* is the node's node ID) in the node's `DataDir`. Node event logs are generated for both management nodes and data nodes.

Node logs are intended to be used only during application development, or for debugging application code.

Both types of event logs can be set to log different subsets of events.

Each reportable event can be distinguished according to three different criteria:

- *Category*: This can be any one of the following values: `STARTUP`, `SHUTDOWN`, `STATISTICS`, `CHECKPOINT`, `NODERESTART`, `CONNECTION`, `ERROR`, or `INFO`.
- *Priority*: This is represented by one of the numbers from 1 to 15 inclusive, where 1 indicates “most important” and 15 “least important.”
- *Severity Level*: This can be any one of the following values: `ALERT`, `CRITICAL`, `ERROR`, `WARNING`, `INFO`, or `DEBUG`.

Both the cluster log and the node log can be filtered on these properties.

The format used in the cluster log is as shown here:

```
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 1: Data usage is 2%(60 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 1: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 1: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 2: Data usage is 2%(76 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 2: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 2: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 3: Data usage is 2%(58 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 3: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 3: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 4: Data usage is 2%(74 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 4: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO      -- Node 4: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 1: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 1: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 2: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 2: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 3: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 3: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO      -- Node 4: Node 9: API version 5.1.15
2007-01-26 19:59:22 [MgmSrvr] ALERT     -- Node 2: Node 7 Disconnected
2007-01-26 19:59:22 [MgmSrvr] ALERT     -- Node 2: Node 7 Disconnected
```

Each line in the cluster log contains the following information:

- A timestamp in `YYYY-MM-DD HH:MM:SS` format.
- The type of node which is performing the logging. In the cluster log, this is always `[MgmSrvr]`.
- The severity of the event.
- The ID of the node reporting the event.
- A description of the event. The most common types of events to appear in the log are connections and disconnections between different nodes in the cluster, and when checkpoints occur. In some cases, the description may contain status information.

16.7.3.1. Logging Management Commands

The following management commands are related to the cluster log:

- `CLUSTERLOG ON`
Turns the cluster log on.
- `CLUSTERLOG OFF`
Turns the cluster log off.
- `CLUSTERLOG INFO`
Provides information about cluster log settings.
- `node_id CLUSTERLOG category=threshold`
Logs *category* events with priority less than or equal to *threshold* in the cluster log.
- `CLUSTERLOG FILTER severity_level`
Toggles cluster logging of events of the specified *severity_level*.

The following table describes the default setting (for all data nodes) of the cluster log category threshold. If an event has a priority with a value lower than or equal to the priority threshold, it is reported in the cluster log.

Note that events are reported per data node, and that the threshold can be set to different values on different nodes.

Category	Default threshold (All data nodes)
STARTUP	7
SHUTDOWN	7
STATISTICS	7
CHECKPOINT	7
NODERESTART	7
CONNECTION	7
ERROR	15
INFO	7

The `STATISTICS` category can provide a great deal of useful data. See [Section 16.7.3.3, “Using CLUSTERLOG STATISTICS”](#), for more information.

Thresholds are used to filter events within each category. For example, a `STARTUP` event with a priority of 3 is not logged unless the threshold for `STARTUP` is changed to 3 or lower. Only events with priority 3 or lower are sent if the threshold is 3.

The following table shows the event severity levels.

Note

These correspond to Unix `syslog` levels, except for `LOG_EMERG` and `LOG_NOTICE`, which are not used or mapped.

1	ALERT	A condition that should be corrected immediately, such as a corrupted system database
2	CRITICAL	Critical conditions, such as device errors or insufficient resources
3	ERROR	Conditions that should be corrected, such as configuration errors
4	WARNING	Conditions that are not errors, but that might require special handling
5	INFO	Informational messages
6	DEBUG	Debugging messages used for NDB Cluster development

Event severity levels can be turned on or off (using `CLUSTERLOG FILTER` — see above). If a severity level is turned on, then all events with a priority less than or equal to the category thresholds are logged. If the severity level is turned off then no events belonging to that severity level are logged.

16.7.3.2. Log Events

An event report reported in the event logs has the following format:

```
datetime [string] severity -- message
```

For example:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

This section discusses all reportable events, ordered by category and severity level within each category.

In the event descriptions, GCP and LCP mean “Global Checkpoint” and “Local Checkpoint”, respectively.

CONNECTION Events

These events are associated with connections between Cluster nodes.

Event	Priority	Severity Level	Description
data nodes connected	8	INFO	Data nodes connected
data nodes disconnected	8	INFO	Data nodes disconnected
Communication closed	8	INFO	SQL node or data node connection closed
Communication opened	8	INFO	SQL node or data node connection opened

CHECKPOINT Events

The logging messages shown here are associated with checkpoints.

Event	Priority	Severity Level	Description
LCP stopped in calc keep GCI	0	ALERT	LCP stopped
Local checkpoint fragment completed	11	INFO	LCP on a fragment has been completed
Global checkpoint completed	10	INFO	GCP finished
Global checkpoint started	9	INFO	Start of GCP: REDO log is written to disk
Local checkpoint completed	8	INFO	LCP completed normally
Local checkpoint started	7	INFO	Start of LCP: data written to disk

STARTUP Events

The following events are generated in response to the startup of a node or of the cluster and of its success or failure. They also provide information relating to the progress of the startup process, including information concerning logging activities.

Event	Priority	Severity Level	Description
Internal start signal received STORRY	15	INFO	Blocks received after completion of restart
New REDO log started	10	INFO	GCI keep <i>X</i> , newest restorable GCI <i>Y</i>
New log started	10	INFO	Log part <i>X</i> , start MB <i>Y</i> , stop MB <i>Z</i>
Node has been refused for inclusion in the cluster	8	INFO	Node cannot be included in cluster due to misconfiguration, inability to establish communication, or other problem
data node neighbors	8	INFO	Shows neighboring data nodes
data node start phase <i>X</i> completed	4	INFO	A data node start phase has been completed
Node has been successfully included into the cluster	3	INFO	Displays the node, managing node, and dynamic ID
data node start phases initiated	1	INFO	NDB Cluster nodes starting
data node all start phases completed	1	INFO	NDB Cluster nodes started
data node shutdown initiated	1	INFO	Shutdown of data node has commenced
data node shutdown aborted	1	INFO	Unable to shut down data node normally

NODERESTART Events

The following events are generated when restarting a node and relate to the success or failure of the node restart process.

Event	Priority	Severity Level	Description
Node failure phase completed	8	ALERT	Reports completion of node failure phases
Node has failed, node state was <i>X</i>	8	ALERT	Reports that a node has failed
Report arbitrator results	2	ALERT	There are eight different possible results for arbitration attempts: - Arbitration check failed — less than 1/2 nodes left - Arbitration check succeeded — node group majority - Arbitration check failed — missing node group - Network partitioning — arbitration required - Arbitration succeeded — affirmative response from node <i>X</i> - Arbitration failed - negative response from node <i>X</i> - Network partitioning - no arbitrator available - Network partitioning - no arbitrator configured
Completed copying a fragment	10	INFO	
Completed copying of dictionary information	8	INFO	
Completed copying distribution information	8	INFO	
Starting to copy fragments	8	INFO	
Completed copying all fragments	8	INFO	

GCP takeover started	7	INFO	
GCP takeover completed	7	INFO	
LCP takeover started	7	INFO	
LCP takeover completed (state = <i>X</i>)	7	INFO	
Report whether an arbitrator is found or not	6	INFO	There are seven different possible outcomes when seeking an arbitrator: • Management server restarts arbitration thread [state=<i>X</i>] • Prepare arbitrator node <i>X</i> [ticket=<i>Y</i>] • Receive arbitrator node <i>X</i> [ticket=<i>Y</i>] • Started arbitrator node <i>X</i> [ticket=<i>Y</i>] • Lost arbitrator node <i>X</i> - process failure [state=<i>Y</i>] • Lost arbitrator node <i>X</i> - process exit [state=<i>Y</i>] • Lost arbitrator node <i>X</i> <error msg> [state=<i>Y</i>]

STATISTICS Events

The following events are of a statistical nature. They provide information such as numbers of transactions and other operations, amount of data sent or received by individual nodes, and memory usage.

Event	Priority	Severity Level	Description
Report job scheduling statistics	9	INFO	Mean internal job scheduling statistics
Sent number of bytes	9	INFO	Mean number of bytes sent to node <i>X</i>
Received # of bytes	9	INFO	Mean number of bytes received from node <i>X</i>
Report transaction statistics	8	INFO	Numbers of: transactions, commits, reads, simple reads, writes, concurrent operations, attribute information, and aborts
Report operations	8	INFO	Number of operations
Report table create	7	INFO	
Memory usage	5	INFO	Data and index memory usage (80%, 90%, and 100%)

ERROR Events

These events relate to Cluster errors and warnings. The presence of one or more of these generally indicates that a major malfunction or failure has occurred.

Event	Priority	Severity	Description
Dead due to missed heartbeat	8	ALERT	Node <i>X</i> declared “dead” due to missed heartbeat
Transporter errors	2	ERROR	
Transporter warnings	8	WARNING	
Missed heartbeats	8	WARNING	Node <i>X</i> missed heartbeat # <i>Y</i>
General warning events	2	WARNING	

INFO Events

These events provide general information about the state of the cluster and activities associated with Cluster maintenance, such as logging and heartbeat transmission.

Event	Priority	Severity	Description
Sent heartbeat	12	INFO	Heartbeat sent to node <i>X</i>
Create log bytes	11	INFO	Log part, log file, MB
General information events	2	INFO	

16.7.3.3. Using **CLUSTERLOG STATISTICS**

The **NDB** management client's **CLUSTERLOG STATISTICS** command can provide a number of useful statistics in its output. The following statistics are reported by the transaction coordinator:

Statistic	Description (Number of...)
Trans. Count	Transactions attempted with this node as coordinator
Commit Count	Transactions committed with this node as coordinator
Read Count	Primary key reads (all)
Simple Read Count	Primary key reads reading the latest committed value
Write Count	Primary key writes (includes all INSERT , UPDATE , and DELETE operations)
AttrInfoCount	Data words used to describe all reads and writes received
Concurrent Operations	All concurrent operations ongoing at the moment the report is taken
Abort Count	Transactions with this node as coordinator that were aborted
Scans	Scans (all)
Range Scans	Index scans

The **ndbd** process has a scheduler that runs in an infinite loop. During each loop scheduler performs the following tasks:

1. Read any incoming messages from sockets into a job buffer.
2. Check whether there are any timed messages to be executed; if so, put these into the job buffer as well.
3. Execute (in a loop) any messages in the job buffer.
4. Send any distributed messages that were generated by executing the messages in the job buffer.
5. Wait for any new incoming messages.

The number of loops executed in the third step is reported as the **Mean Loop Counter**. This statistic increases in size as the utilisation of the TCP/IP buffer improves. You can use this to monitor performance as you add new processes to the cluster.

The **Mean send size** and **Mean receive size** statistics allow you to gauge the efficiency of writes and reads (respectively) between nodes. These values are given in bytes. Higher values mean a lower cost per byte sent or received; the maximum is 64k.

To cause all cluster log statistics to be logged, you can use the following command in the **NDB** management client:

```
ndb_mgm> ALL CLUSTERLOG STATISTICS=15
```

Note

Setting the threshold for **STATISTICS** to 15 causes the cluster log to become very verbose, and to grow quite rapidly in size, in direct proportion to the number of cluster nodes and the amount of activity on the cluster.

16.7.4. Single User Mode

Single user mode allows the database administrator to restrict access to the database system to a single API node, such as a MySQL server (SQL node) or an instance of `ndb_restore`. When entering single user mode, connections to all other API nodes are closed gracefully and all running transactions are aborted. No new transactions are permitted to start.

Once the cluster has entered single user mode, only the designated API node is granted access to the database.

You can use the `ALL STATUS` command to see when the cluster has entered single user mode.

Example:

```
ndb_mgm> ENTER SINGLE USER MODE 5
```

After this command has executed and the cluster has entered single user mode, the API node whose node ID is `5` becomes the cluster's only permitted user.

The node specified in the preceding command must be an API node; attempting to specify any other type of node will be rejected.

Note

When the preceding command is invoked, all transactions running on the designated node are aborted, the connection is closed, and the server must be restarted.

The command `EXIT SINGLE USER MODE` changes the state of the cluster's data nodes from single user mode to normal mode. API nodes — such as MySQL Servers — waiting for a connection (that is, waiting for the cluster to become ready and available), are again permitted to connect. The API node denoted as the single-user node continues to run (if still connected) during and after the state change.

Example:

```
ndb_mgm> EXIT SINGLE USER MODE
```

There are two recommended ways to handle a node failure when running in single user mode:

- Method 1:
 1. Finish all single user mode transactions
 2. Issue the `EXIT SINGLE USER MODE` command
 3. Restart the cluster's data nodes
- Method 2:

Restart database nodes prior to entering single user mode.

16.7.5. Quick Reference: MySQL Cluster SQL Statements

This section discusses several SQL statements that can prove useful in managing and monitoring a MySQL server that is connected to a MySQL Cluster, and in some cases provide information about the cluster itself.

- `SHOW ENGINE NDB STATUS, SHOW ENGINE NDBCLUSTER STATUS`

The output of this statement contains information about the server's connection to the cluster, creation and usage of MySQL Cluster objects, and binary logging for MySQL Cluster replication.

See [Section 12.5.4.13, “SHOW ENGINE Syntax”](#), for a usage example and more detailed information.

- `SHOW ENGINES`

This statement can be used to determine whether or not clustering support is enabled in the MySQL server, and if so, whether it is active.

See [Section 12.5.4.14, “SHOW ENGINES Syntax”](#), for more detailed information.

Note

In MySQL 5.1, this statement no longer supports a [LIKE](#) clause. However, you can use [LIKE](#) to filter queries against the `INFORMATION_SCHEMA.ENGINES`, as discussed in the next item.

- `SELECT * FROM INFORMATION_SCHEMA.ENGINES [WHERE ENGINE LIKE 'NDB%']`

This is the equivalent of `SHOW ENGINES`, but uses the `ENGINES` table of the `INFORMATION_SCHEMA` database (available beginning with MySQL 5.1.5). Unlike the case with the `SHOW ENGINES` statement, it is possible to filter the results using a [LIKE](#) clause, and to select specific columns to obtain information that may be of use in scripts. For example, the following query shows whether the server was built with `NDB` support and, if so, whether it is enabled:

```
mysql> SELECT SUPPORT FROM INFORMATION_SCHEMA.ENGINES
-> WHERE ENGINE LIKE 'NDB%';
+-----+
| support |
+-----+
| ENABLED |
+-----+
```

See [Section 23.18, “The INFORMATION_SCHEMA ENGINES Table”](#), for more information.

- `SHOW VARIABLES LIKE 'NDB%'`

This statement provides a list of most server system variables relating to the `NDB` storage engine, and their values, as shown here:

```
mysql> SHOW VARIABLES LIKE 'NDB%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| ndb_autoincrement_prefetch_sz | 32 |
| ndb_cache_check_time | 0 |
| ndb_extra_logging | 0 |
| ndb_force_send | ON |
| ndb_index_stat_cache_entries | 32 |
| ndb_index_stat_enable | OFF |
| ndb_index_stat_update_freq | 20 |
| ndb_report_thresh_binlog_epoch_slip | 3 |
| ndb_report_thresh_binlog_mem_usage | 10 |
| ndb_use_copying_alter_table | OFF |
| ndb_use_exact_count | ON |
| ndb_use_transactions | ON |
+-----+-----+
```

See [Section 5.2.3, “System Variables”](#), for more information.

- `SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES WHERE VARIABLE_NAME LIKE 'NDB%';`

This statement is the equivalent of the `SHOW` described in the previous item, and provides almost identical output, as shown here:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES
-> WHERE VARIABLE_NAME LIKE 'NDB%';
+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| NDB_AUTOINCREMENT_PREFETCH_SZ | 32 |
| NDB_CACHE_CHECK_TIME | 0 |
| NDB_EXTRA_LOGGING | 0 |
| NDB_FORCE_SEND | ON |
| NDB_INDEX_STAT_CACHE_ENTRIES | 32 |
| NDB_INDEX_STAT_ENABLE | OFF |
| NDB_INDEX_STAT_UPDATE_FREQ | 20 |
| NDB_REPORT_THRESH_BINLOG_EPOCH_SLIP | 3 |
| NDB_REPORT_THRESH_BINLOG_MEM_USAGE | 10 |
| NDB_USE_COPYING_ALTER_TABLE | OFF |
| NDB_USE_EXACT_COUNT | ON |
| NDB_USE_TRANSACTIONS | ON |
+-----+-----+
```

Unlike the case with the `SHOW` statement, it is possible to select individual columns. For example:

```
mysql> SELECT VARIABLE_VALUE
-> FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES
-> WHERE VARIABLE_NAME = 'ndb_force_send';
+-----+
| VARIABLE_VALUE |
+-----+
| ON              |
+-----+
```

See [Section 23.25, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”](#), and [Section 5.2.3, “System Variables”](#), for more information.

- **SHOW STATUS LIKE 'NDB%'**

This statement shows at a glance whether or not the MySQL server is acting as a cluster SQL node, and if so, it provides the MySQL server's cluster node ID, the hostname and port for the cluster management server to which it is connected, and the number of data nodes in the cluster, as shown here:

```
mysql> SHOW STATUS LIKE 'NDB%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_cluster_node_id | 10 |
| Ndb_config_from_host | 192.168.0.103 |
| Ndb_config_from_port | 1186 |
| Ndb_number_of_data_nodes | 4 |
+-----+-----+
```

If the MySQL server was built with clustering support, but it is not connected to a cluster, all rows in the output of this statement contain a zero or an empty string:

```
mysql> SHOW STATUS LIKE 'NDB%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_cluster_node_id | 0 |
| Ndb_config_from_host | 0 |
| Ndb_config_from_port | 0 |
| Ndb_number_of_data_nodes | 0 |
+-----+-----+
```

See also [Section 12.5.4.27, “SHOW STATUS Syntax”](#).

- **SELECT * FROM INFORMATION_SCHEMA.GLOBAL_STATUS WHERE VARIABLE_NAME LIKE 'NDB%';**

Beginning with MySQL 5.1.12, this statement provides similar output to the **SHOW** statement discussed in the previous item. However, unlike the case with **SHOW STATUS**, it is possible using the **SELECT** to extract values in SQL for use in scripts for monitoring and automation purposes.

See [Section 23.24, “The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables”](#), for more information.

16.8. On-line Backup of MySQL Cluster

This section describes how to create a backup and how to restore the database from a backup at a later time.

16.8.1. Cluster Backup Concepts

A backup is a snapshot of the database at a given time. The backup consists of three main parts:

- **Metadata.** The names and definitions of all database tables
- **Table records.** The data actually stored in the database tables at the time that the backup was made
- **Transaction log.** A sequential record telling how and when data was stored in the database

Each of these parts is saved on all nodes participating in the backup. During backup, each node saves these three parts into three files on disk:

- `BACKUP-backup_id.node_id.ctl`

A control file containing control information and metadata. Each node saves the same table definitions (for all tables in the cluster) to its own version of this file.

- `BACKUP-backup_id-0.node_id.data`

A data file containing the table records, which are saved on a per-fragment basis. That is, different nodes save different fragments during the backup. The file saved by each node starts with a header that states the tables to which the records belong. Following the list of records there is a footer containing a checksum for all records.

- `BACKUP-backup_id.node_id.log`

A log file containing records of committed transactions. Only transactions on tables stored in the backup are stored in the log. Nodes involved in the backup save different records because different nodes host different database fragments.

In the listing above, *backup_id* stands for the backup identifier and *node_id* is the unique identifier for the node creating the file.

16.8.2. Using The Management Client to Create a Backup

Before starting a backup, make sure that the cluster is properly configured for performing one. (See [Section 16.8.4, “Configuration for Cluster Backup”](#).)

Creating a backup using the management client involves the following steps:

1. Start the management client (`ndb_mgm`).
2. Execute the command `START BACKUP`.
3. The management client responds as shown here:

```
Waiting for completed, this may take several minutes
Node 1: Backup backup_id started from node management_node_id
```

Here, *backup_id* is the unique identifier for this particular backup. (This identifier will also be saved in the cluster log, if it has not been configured otherwise.) *management_node_id* is the node ID of the management to which the management client is connected.

This means that the cluster has received and processed the backup request. It does *not* mean that the backup has been completed.

Note

Backup messages were not recorded in the cluster log in MySQL 5.1.12 or 5.1.13. The logging of backup operations was restored in MySQL 5.1.14 (see [Bug#24544](#)).

4. When the backup is completed, the management client will indicate this as shown here:

```
Node 1: Backup backup_id started from node management_node_id completed
StartGCP: 417599 StopGCP: 417602
#Records: 105957 #LogRecords: 0
Data: 99719356 bytes Log: 0 bytes
```

The values shown for `StartGCP`, `StopGCP`, `#Records`, `#LogRecords`, `Data`, and `Log` will vary according to the specifics of your cluster.

Cluster backups are created by default in the `BACKUP` subdirectory of the `DataDir` on each data node. This can be overridden for one or more data nodes individually, or for all cluster data nodes in the `config.ini` file using the `BackupDataDir` configuration parameter as discussed in [Identifying Data Nodes](#). The backup files created for a backup with a given *backup_id* are stored in a subdirect-

ory named `BACKUP-backup_id` in the backup directory.

To abort a backup that is already in progress:

1. Start the management client.
2. Execute this command:

```
ndb_mgm> ABORT BACKUP backup_id
```

The number *backup_id* is the identifier of the backup that was included in the response of the management client when the backup was started (in the message `Backup backup_id started from node management_node_id`).

3. The management client will acknowledge the abort request with `Abort of backup backup_id ordered`.

Note

At this point, the management client has not yet received a response from the cluster data nodes to this request, and the backup has not yet actually been aborted.

4. After the backup has been aborted, the management client will report this fact in a manner similar to what is shown here:

```
Node 1: Backup 3 started from 5 has been aborted. Error: 1321 - Backup aborted by user request: Permanent error: User
Node 3: Backup 3 started from 5 has been aborted. Error: 1323 - 1323: Permanent error: Internal error
Node 2: Backup 3 started from 5 has been aborted. Error: 1323 - 1323: Permanent error: Internal error
Node 4: Backup 3 started from 5 has been aborted. Error: 1323 - 1323: Permanent error: Internal error
```

In this example, we have shown sample output for a cluster with 4 data nodes, where the sequence number of the backup to be aborted is `3`, and the management node to which the cluster management client is connected has the node ID `5`. The first node to complete its part in aborting the backup reports that the reason for the abort was due to a request by the user. (The remaining nodes report that the backup was aborted due to an unspecified internal error.)

Note

There is no guarantee that the cluster nodes respond to an `ABORT BACKUP` command in any particular order.

The `Backup backup_id started from node management_node_id has been aborted` messages mean that the backup has been terminated and that all files relating to this backup have been removed from the cluster filesystem.

It is also possible to abort a backup in progress from a system shell using this command:

```
shell> ndb_mgm -e "ABORT BACKUP backup_id"
```

Note

If there is no backup with ID *backup_id* running when an `ABORT BACKUP` is issued, the management client makes no response, nor is it indicated in the cluster log that an invalid abort command was sent.

16.8.3. `ndb_restore` — Restore a Cluster Backup

The cluster restoration program is implemented as a separate command-line utility `ndb_restore`, which can normally be found in the MySQL `bin` directory. This program reads the files created as a result of the backup and inserts the stored information into the database.

`ndb_restore` must be executed once for each of the backup files that were created by the `START BACKUP` command used to create the backup (see [Section 16.8.2, “Using The Management Client to Create a Backup”](#)). This is equal to the number of data nodes in the cluster at the time that the backup was created.

Note

Before using `ndb_restore`, it is recommended that the cluster be running in single user mode, unless you are restoring multiple data nodes in parallel. See [Section 16.7.4, “Single User Mode”](#), for more information about single user mode.

Typical options for this utility are shown here:

```
ndb_restore [-c connectstring] -n node_id [-s] [-m] -b backup_id -r [backup_path=]/path/to/backup/files [-e]
```

The `-c` option is used to specify a connectstring which tells `ndb_restore` where to locate the cluster management server. (See [Section 16.4.4.2, “The Cluster Connectstring”](#), for information on connectstrings.) If this option is not used, then `ndb_restore` attempts to connect to a management server on `localhost:1186`. This utility acts as a cluster API node, and so requires a free connection “slot” to connect to the cluster management server. This means that there must be at least one `[API]` or `[MYSQLD]` section that can be used by it in the cluster `config.ini` file. It is a good idea to keep at least one empty `[API]` or `[MYSQLD]` section in `config.ini` that is not being used for a MySQL server or other application for this reason (see [Section 16.4.4.6, “Defining SQL and Other API Nodes”](#)).

You can verify that `ndb_restore` is connected to the cluster by using the `SHOW` command in the `ndb_mgm` management client. You can also accomplish this from a system shell, as shown here:

```
shell> ndb_mgm -e "SHOW"
```

`-n` is used to specify the node ID of the data node on which the backups were taken.

The first time you run the `ndb_restore` restoration program, you also need to restore the metadata. In other words, you must re-create the database tables — this can be done by running it with the `-m` option. Note that the cluster should have an empty database when starting to restore a backup. (In other words, you should start `ndbd` with `--initial` prior to performing the restore. You should also remove manually any Disk Data files present in the data node's `DataDir`.)

MySQL Cluster 5.1 Carrier Grade Edition

The following information does not apply to users of MySQL Cluster 5.1 Carrier Grade Edition. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#).

It is possible to restore data without restoring table metadata. Prior to MySQL 5.1.17, `ndb_restore` did not perform any checks of table schemas; if a table was altered between the time the backup was taken and when `ndb_restore` was run, `ndb_restore` would still attempt to restore the data to the altered table.

Beginning with MySQL 5.1.17, the default behavior is for `ndb_restore` is to fail with an error if table data do not match the table schema; this can be overridden using the `--skip-table-check` or `-s` option. If this option is used, then `ndb_restore` attempts to fit data into the existing table schema. *The result of restoring a backup to a table schema that does not match the original is unspecified and is subject to change without notice. (Bug#24363)*

The following information applies to all MySQL Cluster users.

The `-b` option is used to specify the ID or sequence number of the backup, and is the same number shown by the management client in the `Backup backup_id completed` message displayed upon completion of a backup. (See [Section 16.8.2, “Using The Management Client to Create a Backup”](#).)

`-e` adds (or restores) epoch information to the cluster replication status table. This is useful for starting replication on a MySQL Cluster replication slave. When this option is used, the row in the `mysql.ndb_apply_status` having `0` in the `id` column is updated if it already exists; such a row is inserted if it does not already exist. (See [Section 16.10.9, “MySQL Cluster Backups With Replication”](#).)

The path to the backup directory is required, and must include the subdirectory corresponding to the ID backup of the backup to be restored. For example, if the data node's `DataDir` is `/var/lib/mysql-cluster`, then the backup directory is `/var/lib/mysql-cluster/BACKUP`, and the backup files for the backup with the ID 3 can be found in `/var/lib/mysql-cluster/BACKUP/BACKUP-3`. The path may be absolute or relative to the directory in which the `ndb_restore` executable is located, and may be optionally prefixed with `backup_path=`.

Important

When restoring cluster backups, you must be sure to restore all data nodes from backups having the same backup ID. Using files from different backups will at best result in restoring the cluster to an inconsistent state, and may fail altogether.

It is possible to restore a backup to a database with a different configuration than it was created from. For example, suppose that a backup with backup ID `12`, created in a cluster with two database nodes having the node IDs `2` and `3`, is to be restored to a cluster with four nodes. Then `ndb_restore` must be run twice — once for each database node in the cluster where the backup was taken. However, `ndb_restore` cannot always restore backups made from a cluster running one version of MySQL to a cluster running a different MySQL version. See [Section 16.5.2, “Cluster Upgrade and Downgrade Compatibility”](#), for more information.

Note

For more rapid restoration, the data may be restored in parallel, provided that there is a sufficient number of cluster connections available. That is, when restoring to multiple nodes in parallel, you must have an `[API]` or `[MYSQLD]` section in the cluster `config.ini` file available for each concurrent `ndb_restore` process. However, the data files must always be applied before the logs.

Most of the options available for this program are shown in the following table:

Long Form	Short Form	Description	Default Value
<code>--backup-id</code>	<code>-b</code>	Backup sequence ID	<code>0</code>
<code>--backup_path</code>	<i>None</i>	Path to backup files	<code>./</code>
<code>--character-sets-dir</code>	<i>None</i>	Specify the directory where character set information can be found	<i>None</i>
<code>--connect</code> , <code>-</code> <code>--connectstring</code> , or <code>-</code> <code>--ndb-connectstring</code>	<code>-c</code> or <code>-C</code>	Set the connectstring in <code>[nodeid=node_id;][host=]host[:port]</code> format	<code>localhost:1186</code>
<code>--core-file</code>	<i>None</i>	Write a core file in the event of an error	<code>TRUE</code>
<code>--debug</code>	<code>-#</code>	Output debug log	<code>d:t:0,/tmp/ndb_restore.trace</code>
<code>-</code> <code>--dont_ignore_systab_0</code>	<code>-f</code>	Do not ignore system table during restore — <i>EXPERIMENTAL; not for production use</i>	<code>FALSE</code>
<code>--help</code> or <code>--usage</code>	<code>-?</code>	Display help message with available options and current values, then exit	[N/A]
<code>--ndb-mgmd-host</code>	<i>None</i>	Set the host and port in <code>host[:port]</code> format for the management server to connect to; this is the same as <code>--connect</code> , <code>--connectstring</code> , or <code>--ndb-connectstring</code> , but without a way to specify the <code>nodeid</code>	<i>None</i>
<code>--ndb-nodegroup-map</code>	<code>-z</code>	Specifies a nodegroup map — <i>Syntax</i> : list of (<code>source_nodegroup</code> , <code>destination_nodegroup</code>)	<i>None</i>
<code>--ndb-nodeid</code>	<i>None</i>	Specify a node ID for the <code>ndb_restore</code> process	<code>0</code>
<code>-</code> <code>-</code> <code>ndb-optimized-node-selection</code>	<i>None</i>	Optimize selection of nodes for transactions	<code>TRUE</code>
<code>--ndb-shm</code>	<i>None</i>	Use shared memory connections when available	<code>FALSE</code>
<code>-</code> <code>-</code> <code>no-restore-disk-objects</code>	<code>-d</code>	Do not restore Disk Data objects such as tablespaces and log file groups	<code>FALSE</code> (in other words, restore Disk Data objects unless this option is used)
<code>--nodeid</code>	<code>-n</code>	Use backup files from node with the specified ID	<code>0</code>
<code>--parallelism</code>	<code>-p</code>	Set from 1 to 1024 parallel transactions to be used during the restoration process	<code>128</code>
<code>--print</code>	<i>None</i>	Print metadata, data, and log to <code>stdout</code>	<code>FALSE</code>
<code>--print_data</code>	<i>None</i>	Print data to <code>stdout</code>	<code>FALSE</code>
<code>--print_log</code>	<i>None</i>	Print log to <code>stdout</code>	<code>FALSE</code>
<code>--print_meta</code>	<i>None</i>	Print metadata to <code>stdout</code>	<code>FALSE</code>
<code>--restore_data</code>	<code>-r</code>	Restore data and logs	<code>FALSE</code>
<code>--restore_epoch</code>	<code>-e</code>	Restore epoch data into the status table; the row in the <code>cluster.apply_status</code> having the id <code>0</code> is inserted or updated as appropriate — this is convenient when starting up replication on a MySQL Cluster replication slave	<code>FALSE</code>

<code>--restore-meta</code>	<code>-m</code>	Restore table metadata	FALSE
<code>--skip-table-check</code>	<code>-s</code>	Do not check table schemas (Added in MySQL 5.1.17; <i>not supported in MySQL Cluster 5.1 Carrier Grade Edition</i>)	FALSE
<code>--version</code>	<code>-V</code>	Output version information and exit	[N/A]

MySQL Cluster 5.1 Carrier Grade Edition

The following information does not apply to users of MySQL Cluster 5.1 Carrier Grade Edition. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

Beginning with MySQL 5.1.18, several additional options are available for use with the `--print_data` option in generating data dumps, either to `stdout`, or to a file. These are similar to some of the options used with `mysqldump`, and are shown in the following table:

Long Form	Short Form	Description	Default Value
<code>--tab</code>	<code>-T</code>	Creates dumpfiles, one per table, each named <code>tbl_name.txt</code> . Takes as its argument the path to the directory where the files should be saved (required; use <code>.</code> for the current directory).	None
<code>--fields-enclosed-by</code>	None	String used to enclose all column values	None
<code>--fields-optionally-enclosed-by</code>	None	String used to enclose column values containing character data (such as <code>CHAR</code> , <code>VARCHAR</code> , <code>BINARY</code> , <code>TEXT</code> , or <code>ENUM</code>)	None
<code>--fields-terminated-by</code>	None	String used to separate column values	<code>\t</code> (tab character)
<code>--hex</code>	None	Use hex format for binary values	[N/A]
<code>--lines-terminated-by</code>	None	String used to terminate each line	<code>\n</code> (linefeed character)
<code>--append</code>	None	When used with <code>--tab</code> , causes the data to be appended to existing files of the same name	[N/A]

Note

If a table has no explicit primary key, then the output generated when using the `--print` includes the table's hidden primary key.

Beginning with MySQL 5.1.18, it is possible to restore selected databases, or to restore selected tables from a given database using the syntax shown here:

```
ndb_restore other_options db_name_1 [db_name_2[, db_name_3][, ...] | tbl_name_1[, tbl_name_2][, ...]]
```

In other words, you can specify either of the following to be restored:

- All tables from one or more databases
- One or more tables from a single database

The following information applies to all MySQL Cluster users.

Note

`ndb_restore` reports both temporary and permanent errors. In the case of temporary errors, it may be able to recover from them. Beginning with MySQL 5.1.12, it reports `Restore successful, but encountered temporary error, please look at configuration` in such cases.

16.8.4. Configuration for Cluster Backup

Five configuration parameters are essential for backup:

- **BackupDataBufferSize**
The amount of memory used to buffer data before it is written to disk.
- **BackupLogBufferSize**
The amount of memory used to buffer log records before these are written to disk.
- **BackupMemory**
The total memory allocated in a database node for backups. This should be the sum of the memory allocated for the backup data buffer and the backup log buffer.
- **BackupWriteSize**
The default size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.
- **BackupMaxWriteSize**
The maximum size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

More detailed information about these parameters can be found in [Backup Parameters](#).

16.8.5. Backup Troubleshooting

If an error code is returned when issuing a backup request, the most likely cause is insufficient memory or disk space. You should check that there is enough memory allocated for the backup.

Important

If you have set **BackupDataBufferSize** and **BackupLogBufferSize** and their sum is greater than 4MB, then you must also set **BackupMemory** as well. See **BackupMemory**.

You should also make sure that there is sufficient space on the hard drive partition of the backup target.

NDB does not support repeatable reads, which can cause problems with the restoration process. Although the backup process is “hot”, restoring a MySQL Cluster from backup is not a 100% “hot” process. This is due to the fact that, for the duration of the restore process, running transactions get non-repeatable reads from the restored data. This means that the state of the data is inconsistent while the restore is in progress.

MySQL Enterprise

MySQL Enterprise subscribers will find more information about Cluster backup in the Knowledge Base article, [How Do I Backup my Cluster Database](#). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

16.9. Cluster Utility Programs

This section discusses the MySQL Cluster utility programs that can be found in the `mysql/bin` directory. Each of these — except for `ndb_size.pl` and `ndb_error_reporter` — is a standalone binary that can be used from a system shell, and that does not need to connect to a MySQL server (nor even requires that a MySQL server be connected to the cluster).

These utilities can also serve as examples for writing your own applications using the **NDB** API. The source code for most of these programs may be found in the `storage/ndb/tools` directory of the MySQL 5.1 tree (see [Section 2.9](#), “MySQL Installation Using a

Source Distribution”). The **NDB** API is not covered in this manual; please refer to the [NDB API Guide](#) for information about this API.

All of the **NDB** utilities are listed here with brief descriptions:

- **ndb_config**: Retrieves Cluster configuration option values.
- **ndb_cpcd**: Used in testing and debugging MySQL Cluster.
- **ndb_delete_all**: Deletes all rows from a given table.
- **ndb_desc**: Lists all properties of an **NDB** table.
- **ndb_drop_index**: Drops the specified index from an **NDB** table.
- **ndb_drop_table**: Drops an **NDB** table.
- **ndb_error_reporter**: Can be used to gather information useful for diagnosing problems with the cluster.
- **ndb_mgm**: This is the MySQL Cluster management client, which is discussed in [Section 16.7.2, “Commands in the MySQL Cluster Management Client”](#).
- **ndb_print_backup_file**: Prints diagnostic information obtained from cluster backup files.
- **ndb_print_schema_file**: Prints diagnostic information obtained from cluster schema files.
- **ndb_print_sys_file**: Prints diagnostic information obtained from cluster system files.
- **ndb_restore**: This utility is used to restore a cluster from backup. See [Section 16.8.3, “ndb_restore — Restore a Cluster Backup”](#), for more information.
- **ndb_select_all**: Prints all rows from an **NDB** table.
- **ndb_select_count**: Gets the number of rows in one or more **NDB** tables.
- **ndb_show_tables**: Shows all **NDB** tables anywhere in the cluster.
- **ndb_size.pl**: Examines all the tables in a given non-Cluster database and calculates the amount of storage each would require if it were converted to use the **NDB** storage engine.
- **ndb_waiter**: Reports on the status of cluster data nodes in a manner similar to that of the management client command **ALL STATUS**.
-

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#).

ndbd_redo_log_reader: Reads a redo log file, checking it for errors, printing it in a human-readable format, or both.

Note

An alpha version of this utility was made available in MySQL 5.1.15-ndb-6.1.3. Currently, it should be considered experimental.

The following information applies to all MySQL Cluster users.

Most of these utilities need to connect to a Cluster management server in order to function. The exceptions are **ndb_size.pl** (see below), and the following utilities which access a cluster data node filesystem and so need to be run on a data node host:

- **ndb_print_backup_file**
- **ndb_print_schema_file**

- `ndb_print_sys_file`
- `ndbd_redo_log_reader` (*MySQL Cluster 5.1 Carrier Grade Edition only*)

`ndb_size.pl` is a Perl script which is also intended to be used from the shell; however it is a MySQL application and must be able to connect to a MySQL server. See [Section 16.9.15, “ndb_size.pl — NDBCluster Size Requirement Estimator”](#), for additional requirements for using this script.

`ndb_error_reporter` is also a Perl script. It is used to gather cluster data node and management node logs together into a tarball to submit along with a bug report. It can use `ssh` or `scp` to access the node filesystems remotely.

Additional information about each of these utilities (except for `ndb_mgm` and `ndb_restore`) can be found in the sections that follow.

Note

All of these utilities (except for `ndb_size.pl` and `ndb_config`) can use the options discussed in [Section 16.6.5, “Command Options for MySQL Cluster Processes”](#). Additional options specific to each utility program are discussed in the individual program listings.

The order in which these options are used is generally not important. For example, all of these commands produce exactly the same output:

- `ndb_desc -c localhost fish -d test`
- `ndb_desc fish -c localhost -d test`
- `ndb_desc -d test fish -c localhost`

16.9.1. `ndb_config` — Extract NDB Configuration Information

This tool extracts configuration information for data nodes, SQL nodes, and API nodes from a cluster management node (and possibly its `config.ini` file).

Usage:

```
ndb_config options
```

The *options* available for this utility differ somewhat from those used with the other utilities, and so are listed in their entirety in the next section, followed by some examples.

Options:

- `--usage`, `--help`, or `-?`

Causes `ndb_config` to print a list of available options, and then exit.

- `--version`, `-V`

Causes `ndb_config` to print a version information string, and then exit.

- `--ndb-connectstring=connect_string`

Specifies the connectstring to use in connecting to the management server. The format for the connectstring is the same as described in [Section 16.4.4.2, “The Cluster Connectstring”](#), and defaults to `localhost:1186`.

The use of `-c` as a short version for this option is supported for `ndb_config` beginning with MySQL 5.1.12.

- `--config-file=path-to-file`

Gives the path to the management server's configuration file (`config.ini`). This may be a relative or absolute path. If the management node resides on a different host from the one on which `ndb_config` is invoked, then an absolute path must be used.

- `--query=query-options, -q query-options`

This is a comma-delimited list of *query options* — that is, a list of one or more node attributes to be returned. These include `id` (node ID), type (node type — that is, `ndbd`, `mysqld`, or `ndb_mgmd`), and any configuration parameters whose values are to be obtained.

For example, `--query=id,type,indexmemory,datamemory` would return the node ID, node type, `DataMemory`, and `IndexMemory` for each node.

Note

If a given parameter is not applicable to a certain type of node, then an empty string is returned for the corresponding value. See the examples later in this section for more information.

- `--host=hostname`

Specifies the hostname of the node for which configuration information is to be obtained.

- `--id=node_id, --nodeid=node_id`

Used to specify the node ID of the node for which configuration information is to be obtained.

- `--nodes`

(Tells `ndb_config` to print information from parameters defined in `[ndbd]` sections only. Currently, using this option has no affect, since these are the only values checked, but it may become possible in future to query parameters set in `[tcp]` and other sections of cluster configuration files.)

- `--type=node_type`

Filters results so that only configuration values applying to nodes of the specified *node_type* (`ndbd`, `mysqld`, or `ndb_mgmd`) are returned.

- `--fields=delimiter, -f delimiter`

Specifies a *delimiter* string used to separate the fields in the result. The default is “,” (the comma character).

Note

If the *delimiter* contains spaces or escapes (such as `\n` for the linefeed character), then it must be quoted.

- `--rows=separator, -r separator`

Specifies a *separator* string used to separate the rows in the result. The default is a space character.

Note

If the *separator* contains spaces or escapes (such as `\n` for the linefeed character), then it must be quoted.

Examples:

1. To obtain the node ID and type of each node in the cluster:

```
shell> ./ndb_config --query=id,type --fields=':' --rows='\n'
1:ndbd
2:ndbd
3:ndbd
4:ndbd
5:ndb_mgmd
6:mysqld
7:mysqld
8:mysqld
9:mysqld
```

In this example, we used the `--fields` options to separate the ID and type of each node with a colon character (:), and the `--rows` options to place the values for each node on a new line in the output.

- To produce a connectstring that can be used by data, SQL, and API nodes to connect to the management server:

```
shell> ./ndb_config --config-file=/usr/local/mysql/cluster-data/config.ini --query=hostname,portnumber --fields=: --row
192.168.0.179:1186
```

- This invocation of `ndb_config` checks only data nodes (using the `--type` option), and shows the values for each node's ID and hostname, and its `DataMemory`, `IndexMemory`, and `DataDir` parameters:

```
shell> ./ndb_config --type=ndbd --query=id,host,databmemory,indexmemory,datadir -f ' : ' -r '\n'
1 : 192.168.0.193 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
2 : 192.168.0.112 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
3 : 192.168.0.176 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
4 : 192.168.0.119 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
```

In this example, we used the short options `-f` and `-r` for setting the field delimiter and row separator, respectively.

- To exclude results from any host except one in particular, use the `--host` option:

```
shell> ./ndb_config --host=192.168.0.176 -f : -r '\n' -q id,type
3:ndbd
5:ndb_mgmd
```

In this example, we also used the short form `-q` to determine the attributes to be queried.

Similarly, you can limit results to a node with a specific ID using the `--id` or `--nodeid` option.

16.9.2. `ndb_cpcd` — Automate Testing for NDB Development

This utility is found in the `libexec` directory. It is part of an internal automated test framework used in testing and debugging MySQL Cluster. Because it can control processes on remote systems, it is not advisable to use `ndb_cpcd` in a production cluster.

Because some users may be interested in employing the Cluster testing framework for their own development or testing purposes, we intend to make details of this application's usage available in the near future as part of the MySQL Internals Manual.

The source files for `ndb_cpcd` may be found in the directory `storage/ndb/src/cw/cpcd`, in the MySQL 5.1 source tree.

16.9.3. `ndb_delete_all` — Delete All Rows from NDB Table

`ndb_delete_all` deletes all rows from the given NDB table. In some cases, this can be much faster than `DELETE` or even `TRUNCATE`.

Usage:

```
ndb_delete_all -c connect_string tbl_name -d db_name
```

This deletes all rows from the table named `tbl_name` in the database named `db_name`. It is exactly equivalent to executing `TRUNCATE db_name.tbl_name` in MySQL.

Additional Options:

- `--transactional, -t`

Use of this option causes the delete operation to be performed as a single transaction.

Warning

With very large tables, using this option may cause the number of operations available to the cluster to be exceeded.

16.9.4. `ndb_desc` — Describe NDB Tables

`ndb_desc` provides a detailed description of one or more NDB tables.

Usage:

```
ndb_desc -c connect_string tbl_name -d db_name
```

Sample Output:

MySQL table creation and population statements:

```
USE test;

CREATE TABLE fish (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name VARCHAR(20),

  PRIMARY KEY pk (id),
  UNIQUE KEY uk (name)
) ENGINE=NDBCLUSTER;

INSERT INTO fish VALUES
  ('','guppy'), ('','tuna'), ('','shark'),
  ('','manta ray'), ('','grouper'), ('','puffer');
```

Output from `ndb_desc`:

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 16777221
Fragment type: 5
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 2
Number of primary keys: 1
Length of frm data: 268
Row Checksum: 1
Row GCI: 1
TableStatus: Retrieved
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
name Varchar(20;latin1_swedish_ci) NULL AT=SHORT_VAR ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
uk(name) - OrderedIndex
PRIMARY(id) - OrderedIndex
uk$unique(name) - UniqueHashIndex

-- Per partition info --
Partition Row count Commit count Frag fixed memory Frag var sized memory
2          2          2          65536          327680
1          2          2          65536          327680
3          2          2          65536          327680

NDBT_ProgramExit: 0 - OK
```

Additional Options:

- `--extra-partition-info, -p`

Prints additional information about the table's partitions.

- Information about multiple tables can be obtained in a single invocation of `ndb_desc` by using their names, separated by spaces. All of the tables must be in the same database.

16.9.5. `ndb_drop_index` — Drop Index from NDB Table

`ndb_drop_index` drops the specified index from an NDB table. *It is recommended that you use this utility only as an example for writing NDB API applications* — see the Warning later in this section for details.

Usage:

```
ndb_drop_index -c connect_string table_name index -d db_name
```

The statement shown above drops the index named *index* from the *table* in the *database*.

Additional Options: None that are specific to this application.

Warning

Operations performed on Cluster table indexes using the NDB API are not visible to MySQL and make the table unusable by a MySQL server. If you use this program to drop an index, then try to access the table from an SQL node, an error results, as shown here:

```
shell> ./ndb_drop_index -c localhost dogs ix -d ctest1
Dropping index dogs/idx...OK

NDBT_ProgramExit: 0 - OK

shell> ./mysql -u jon -p ctest1
Enter password: *****
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 5.1.12-beta-20060817

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW TABLES;
+-----+
| Tables_in_ctest1 |
+-----+
| a                 |
| bt1               |
| bt2               |
| dogs              |
| employees         |
| fish              |
+-----+
6 rows in set (0.00 sec)

mysql> SELECT * FROM dogs;
ERROR 1296 (HY000): GOT ERROR 4243 'INDEX NOT FOUND' FROM NDBCLUSTER
```

In such a case, your *only* option for making the table available to MySQL again is to drop the table and re-create it. You can use either the SQL statement `DROP TABLE` or the `ndb_drop_table` utility (see [Section 16.9.6, “ndb_drop_table — Drop NDB Table”](#)) to drop the table.

16.9.6. ndb_drop_table — Drop NDB Table

`ndb_drop_table` drops the specified NDB table. (If you try to use this on a table created with a storage engine other than NDB, it fails with the error `723: NO SUCH TABLE EXISTS`.) This operation is extremely fast — in some cases, it can be an order of magnitude faster than using `DROP TABLE` on an NDB table from MySQL.

Usage:

```
ndb_drop_table -c connect_string tbl_name -d db_name
```

Additional Options: None.

16.9.7. ndb_error_reporter — NDB Error-Reporting Utility

`ndb_error_reporter` creates an archive from data node and management node log files that can be used to help diagnose bugs or other problems with a cluster. *It is highly recommended that you make use of this utility when filing reports of bugs in MySQL Cluster.*

Usage:

```
ndb_error_reporter path/to/config-file [username] [--fs]
```

This utility is intended for use on a management node host, and requires the path to the management host configuration file (`config.ini`). Optionally, you can supply the name of a user that is able to access the cluster's data nodes via SSH, in order to copy the data node log files. `ndb_error_reporter` then includes all of these files in archive that is created in the same directory in which it is run. The archive is named `ndb_error_report_YYYYMMDDHHMMSS.tar.bz2`, where `YYYYMMDDHHMMSS` is a datetime string.

If the `--fs` is used, then the data node filesystems are also copied to the management host and included in the archive that is produced by this script. As data node filesystems can be extremely large even after being compressed, we ask that you please do *not* send archives created using this option to MySQL AB unless you are specifically requested to do so.

16.9.8. `ndb_print_backup_file` — Print NDB Backup File Contents

`ndb_print_backup_file` obtains diagnostic information from a cluster backup file.

Usage:

```
ndb_print_backup_file file_name
```

file_name is the name of a cluster backup file. This can be any of the files (`.Data`, `.ctl`, or `.log` file) found in a cluster backup directory. These files are found in the data node's backup directory under the subdirectory `BACKUP-#`, where `#` is the sequence number for the backup. For more information about cluster backup files and their contents, see [Section 16.8.1, “Cluster Backup Concepts”](#).

Like `ndb_print_schema_file` and `ndb_print_sys_file` (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_backup_file` must be run on a cluster data node, since it accesses the data node filesystem directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options: None.

16.9.9. `ndb_print_schema_file` — Print NDB Schema File Contents

`ndb_print_schema_file` obtains diagnostic information from a cluster schema file.

Usage:

```
ndb_print_schema_file file_name
```

file_name is the name of a cluster schema file.

Like `ndb_print_backup_file` and `ndb_print_sys_file` (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) `ndb_schema_backup_file` must be run on a cluster data node, since it accesses the data node filesystem directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options: None.

16.9.10. `ndb_print_sys_file` — Print NDB System File Contents

`ndb_print_sys_file` obtains diagnostic information from a cluster system file.

Usage:

```
ndb_print_sys_file file_name
```

file_name is the name of a cluster system file (sysfile). Cluster system files are located in a data node's data directory (`DataDir`); the path under this directory to system files matches the pattern `ndb_#_fs/D#/DBDIH/P#.sysfile`. In each case, the `#` represents a number (not necessarily the same number).

Like `ndb_print_backup_file` and `ndb_print_schema_file` (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_backup_file` must be run on a cluster data node, since it accesses the data node filesystem directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options: None.

16.9.11. `ndbd_redo_log_reader` — Check and Print Content of Cluster Redo Log

MySQL Cluster 5.1 Carrier Grade Edition

The information in this section applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

Reads a redo log file, checking it for errors, printing its contents in a human-readable format, or both. `ndbd_redo_log_reader` is intended for use primarily by MySQL developers and support personnel in debugging and diagnosing problems.

This utility was made available as part of the default builds of MySQL Cluster 5.1 Carrier Grade Edition beginning with MySQL 5.1.15-ndb-6.1.3. It remains under development, and its syntax and behavior are subject to change in future releases. For this reason, it should be considered experimental at this time.

The C++ source files for `ndbd_redo_log_reader` can be found in the directory `/storage/ndb/src/kernel/blocks/dblqh/redoLogReader`.

Usage:

```
ndbd_redo_log_reader file_name [options]
```

file_name is the name of a cluster REDO log file. REDO log files are located in the numbered directories under the data node's data directory (`DataDir`); the path under this directory to the REDO log files matches the pattern `ndb_#_fs/D#/LCP/#/T#F#.Data`. In each case, the `#` represents a number (not necessarily the same number).

Additional Options:

The name of the file to be read may be followed by one or more of the options listed here:

- `-noprint`: Do not print the contents of the log file.
- `-nocheck`: Do not check the log file for errors.

Like `ndb_print_backup_file` and `ndb_print_schema_file` (and unlike most of the NDB utilities that are intended to be run on a management server host or to connect to a management server) `ndbd_redo_log_reader` must be run on a cluster data node, since it accesses the data node filesystem directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

16.9.12. `ndb_select_all` — Print Rows from NDB Table

`ndb_select_all` prints all rows from an NDB table to `stdout`.

Usage:

```
ndb_select_all -c connect_string tbl_name -d db_name [> file_name]
```

Additional Options:

- `--lock=lock_type, -l lock_type`

Employs a lock when reading the table. Possible values for *lock_type* are:

- `0`: Read lock
- `1`: Read lock with hold
- `2`: Exclusive read lock

There is no default value for this option.

- `--order=index_name, -o index_name`

Orders the output according to the index named *index_name*. Note that this is the name of an index, not of a column, and that the index must have been explicitly named when created.

- `--descending, -z`
Sorts the output in descending order. This option can be used only in conjunction with the `-o (--order)` option.
- `--header=FALSE`
Excludes column headers from the output.
- `--useHexFormat -x`
Causes all numeric values to be displayed in hexadecimal format. This does not affect the output of numerals contained in strings or datetime values.
- `--delimiter=character, -D character`
Causes the *character* to be used as a column delimiter. Only table data columns are separated by this delimiter.
The default delimiter is the tab character.
- `--disk`
Adds a disk reference column to the output. The column is non-empty only for Disk Data tables having non-indexed columns.
- `--rowid`
Adds a `ROWID` column providing information about the fragments in which rows are stored.
- `--gci`
Adds a column to the output showing the global checkpoint at which each row was last updated. See [Section 16.15, “MySQL Cluster Glossary”](#), and [Section 16.7.3.2, “Log Events”](#), for more information about checkpoints.
- `--tupscan, -t`
Scan the table in the order of the tuples.
- `--nodata`
Causes any table data to be omitted.

Sample Output:

Output from a MySQL `SELECT` statement:

```
mysql> SELECT * FROM ctest1.fish;
+----+-----+
| id | name |
+----+-----+
| 3  | shark|
| 6  | puffer|
| 2  | tuna |
| 4  | manta ray|
| 5  | grouper|
| 1  | guppy|
+----+-----+
6 rows in set (0.04 sec)
```

Output from the equivalent invocation of `ndb_select_all`:

```
shell> ./ndb_select_all -c localhost fish -d ctest1
id      name
3       [shark]
6       [puffer]
2       [tuna]
4       [manta ray]
5       [grouper]
1       [guppy]
6 rows returned
NDBT_ProgramExit: 0 - OK
```

Note that all string values are enclosed by square brackets (“[...]”) in the output of `ndb_select_all`. For a further example, consider the table created and populated as shown here:

```
CREATE TABLE dogs (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name VARCHAR(25) NOT NULL,
  breed VARCHAR(50) NOT NULL,
  PRIMARY KEY pk (id),
  KEY ix (name)
)
TABLESPACE ts STORAGE DISK
ENGINE=NDB;

INSERT INTO dogs VALUES
  ('', 'Lassie', 'collie'),
  ('', 'Scooby-Doo', 'Great Dane'),
  ('', 'Rin-Tin-Tin', 'Alsatian'),
  ('', 'Rosscoe', 'Mutt');
```

This demonstrates the use of several additional `ndb_select_all` options:

```
shell> ./ndb_select_all -d ctest1 dogs -o ix -z --gci --disk
GCI      id name      breed      DISK_REF
834461   2 [Scooby-Doo] [Great Dane] [ m_file_no: 0 m_page: 98 m_page_idx: 0 ]
834878   4 [Rosscoe]   [Mutt]      [ m_file_no: 0 m_page: 98 m_page_idx: 16 ]
834463   3 [Rin-Tin-Tin] [Alsatian] [ m_file_no: 0 m_page: 34 m_page_idx: 0 ]
835657   1 [Lassie]    [Collie]    [ m_file_no: 0 m_page: 66 m_page_idx: 0 ]
4 rows returned

NDBT_ProgramExit: 0 - OK
```

16.9.13. `ndb_select_count` — Print Row Counts for NDB Tables

`ndb_select_count` prints the number of rows in one or more NDB tables. With a single table, the result is equivalent to that obtained by using the MySQL statement `SELECT COUNT(*) FROM tbl_name`.

Usage:

```
ndb_select_count [-c connect_string] -ddb_name tbl_name [, tbl_name2 [, ...]]
```

Additional Options: None that are specific to this application. However, you can obtain row counts from multiple tables in the same database by listing the table names separated by spaces when invoking this command, as shown under **Sample Output**.

Sample Output:

```
shell> ./ndb_select_count -c localhost -d ctest1 fish dogs
6 records in table fish
4 records in table dogs

NDBT_ProgramExit: 0 - OK
```

16.9.14. `ndb_show_tables` — Display List of NDB Tables

`ndb_show_tables` displays a list of all NDB database objects in the cluster. By default, this includes not only both user-created tables and NDB system tables, but NDB-specific indexes, internal triggers, and Cluster Disk Data objects as well.

Usage:

```
ndb_show_tables [-c connect_string]
```

Additional Options:

- `--loops, -l`

Specifies the number of times the utility should execute. This is 1 when this option is not specified, but if you do use the option, you must supply an integer argument for it.

- `--parsable, -p`

Using this option causes the output to be in a format suitable for use with [LOAD DATA INFILE](#).

- `--type, -t`

Can be used to restrict the output to one type of object, specified by an integer type code as shown here:

- 1: System table
- 2: User-created table
- 3: Unique hash index

Any other value causes all [NDB](#) database objects to be listed (the default).

- `--unqualified, -u`

If specified, this causes unqualified object names to be displayed.

Note

Only user-created Cluster tables may be accessed from MySQL; system tables such as [SYSTAB_0](#) are not visible to [mysqld](#). However, you can examine the contents of system tables using [NDB](#) API applications such as [ndb_select_all](#) (see [Section 16.9.12, “ndb_select_all — Print Rows from NDB Table”](#)).

16.9.15. [ndb_size.pl](#) — NDBCluster Size Requirement Estimator

This is a Perl script that can be used to estimate the amount of space that would be required by a MySQL database if it were converted to use the [NDBCluster](#) storage engine. Unlike the other utilities discussed in this section, it does not require access to a MySQL Cluster (in fact, there is no reason for it to do so). However, it does need to access the MySQL server on which the database to be tested resides.

Requirements:

- A running MySQL server. The server instance does not have to provide support for MySQL Cluster.
- A working installation of Perl.
- The [DBI](#) module, which can be obtained from CPAN if it is not already part of your Perl installation. (Many Linux and other operating system distributions provide their own packages for this library.)

Note

Previous to MySQL 5.1.18, [ndb_size.pl](#) also required the [HTML::Template](#) module.

- The [ndb_size.tmpl](#) template file, which you should be able to find in the [share/mysql](#) directory of your MySQL installation. This file should be copied or moved into the same directory as [ndb_size.pl](#) — if it is not there already — before running the script.
- A MySQL user account having the necessary privileges. If you do not wish to use an existing account, then creating one using [GRANT USAGE ON db_name.*](#) — where [db_name](#) is the name of the database to be examined — is sufficient for this purpose.

[ndb_size.pl](#) and [ndb_size.tmpl](#) can also be found in the MySQL sources in [storage/ndb/tools](#). If these files are not present in your MySQL installation, you can obtain them from the [MySQLForge project page](#).

Usage:

```
perl ndb_size.pl db_name hostname username password > file_name.html
```

The command shown connects to the MySQL server at [hostname](#) using the account of the user [username](#) having the password [password](#), analyses all of the tables in database [db_name](#), and generates a report in HTML format which is directed to the file

file_name.html. (Without the redirection, the output is sent to *stdout*.) This figure shows partial sample output as viewed in a Web browser:

MySQL Cluster analysis for world

This is an automated analysis of the DBI:mysql:database=world;host=192.168.0.176 database for migration into MySQL Cluster. No warranty is made to the accuracy of the information.

This information should be valid for MySQL 4.1 and 5.0. Since 5.1 is not a final release yet, the numbers should be used as a guide only.

Parameter Settings

NOTE the configuration parameters below do not take into account system tables and other requirements.

Parameter	4.1	5.0	5.1
DataMemory (kb)	544	544	544
IndexMemory (kb)	192	136	136
MaxNoOfTables	3	3	3
MaxNoOfAttributes	24	24	24
MaxNoOfOrderedIndexes	3	3	3
MaxNoOfUniqueHashIndexes	3	3	3
MaxNoOfTriggers	12	12	12

Memory usage because of parameters

Usage is in kilobytes. Actual usage will vary as you should set the parameters larger than those listed in the table above.

Parameter	4.1	5.0	5.1
Attributes	5	5	5
Tables	60	60	60
OrderedIndexes	30	30	30
UniqueHashIndexes	45	45	45

Table List

- [City](#)
- [Country](#)
- [CountryLanguage](#)

City

Column	Type	Size	Key	4.1 NDB Size	5.0 NDB Size	5.1 NDB Size
ID	int	11	PRI	4	4	4
District	char	20		20	20	20
Name	char	35		36	36	36
Population	int	11		4	4	4
CountryCode	char	3		4	4	4

Indexes

We assume that indexes are ORDERED (not created USING HASH). If order is not required, 10 bytes of data memory can be saved per row if the index is created USING HASH

Index	Type	Columns	4.1 IdxMem	5.0 IdxMem	5.1 IdxMem	4.1 DatMem	5.0 DatMem	5.1 DatMem
PRIMARY	BTREE	ID	29	25	25	10	10	10

DataMemory Usage

	4.1	5.0	5.1
Row Overhead	16	16	16
Column DataMemory/Row	68	68	68
Index DataMemory/Row	10	10	10
Total DataMemory/Row	94	94	94
Rows per 32 kb page	347	347	347
Current number of rows	4079	4079	4079
Total DataMemory (kb)	384	384	384

IndexMemory Usage

	4.1	5.0	5.1
IndexMemory/Row	29	25	25
Rows per 8kb page	282	327	327
Current number of rows	4079	4079	4079
Total IndexMemory (kb)	120	104	104

The output from this script includes:

- Minimum values for the `DataMemory`, `IndexMemory`, `MaxNoOfTables`, `MaxNoOfAttributes`, `MaxNoOfOrderedIndexes`, `MaxNoOfUniqueHashIndexes`, and `MaxNoOfTriggers` configuration parameters required to accommodate the tables analysed.
- Memory requirements for all of the tables, attributes, ordered indexes, and unique hash indexes defined in the database.
- The `IndexMemory` and `DataMemory` required per table and table row.

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

Beginning with MySQL 5.1.22-ndb-6.2.5, the behavior of this utility and its available options have changed. ([Bug#28683](#), [Bug#28253](#))

Typical usage is shown here:

```
ndb_size.pl [--database=db_name|ALL] [--hostname=host[:port]] [--socket=socket] [--user=user] [--password=password] [--help]
```

Important

`ndb_size.pl` now takes named options, each of which is optional.

By default, this utility attempts to analyze all databases on the server. You can specify a single database using the `--database` option; the default behavior can be made explicit by using `ALL` for the name of the database. You can also exclude one or more databases by using the `--excludedbs` with a comma-separated list of the names of the databases to be skipped. Similarly, you can cause specific tables to be skipped by listing their names, separated by commas, following the optional `--excludetables` option. A host-name (and possibly a port as well) can be specified using `--hostname`; the default is `localhost:3306`. If necessary, you can specify a socket; the default is `/var/lib/mysql.sock`. A MySQL username and password can be specified the corresponding options shown. It also possible to control the format of the output using the `--format` option; this can take either of the values `html` or `text`, with `text` being the default. An example of the text output is shown here:

```
shell> ndb_size.pl --database=test --socket=/tmp/mysql.sock
ndb_size.pl report for database: 'test' (1 tables)
-----
Connected to: DBI:mysql:host=localhost;mysql_socket=/tmp/mysql.sock
Including information for versions: 4.1, 5.0, 5.1

test.t1
-----
DataMemory for Columns (* means var sized DataMemory):
  Column Name      Type      Var sized  Key  4.1  5.0  5.1
  -----
  HIDDEN_NDB_PKEY  bigint
  c2               varchar(50)  Y    PRI  8    8    8
  c1               int(11)      52   52   4*
  4               4         4    4
  --             --         --    --
  Fixed Size Columns DM/Row  64    64    12
  Var size Columns DM/Row   0     0     4

DataMemory for Indexes:
  Index Name      Type      4.1  5.0  5.1
  -----
  PRIMARY        BTREE    16   16   16
  --             --     --   --   --
  Total Index DM/Row      16   16   16

IndexMemory for Indexes:
  Index Name      4.1  5.0  5.1
  -----
  PRIMARY        33   16   16
  --             --   --   --
  Indexes IM/Row   33   16   16

Summary (for THIS table):
  4.1  5.0  5.1
  -----
  Fixed Overhead DM/Row  12   12   16
  NULL Bytes/Row        4     4     4
  DataMemory/Row       96   96   48 (Includes overhead, bitmap and indexes)
  Var size Overhead DM/Row  0     0     8
  Var size NULL Bytes/Row  0     0     4
  Avg Var side DM/Row     0     0    16
```

No. Rows	0	0	0	
Rows/32kb DM Page	340	340	680	
Fixedsize DataMemory (KB)	0	0	0	
Rows/32kb Varsize DM Page	0	0	2040	
Varsize DataMemory (KB)	0	0	0	
Rows/8kb IM Page	248	512	512	
IndexMemory (KB)	0	0	0	
Parameter Minimum Requirements				

* indicates greater than default				
Parameter	Default	4.1	5.0	5.1
DataMemory (KB)	81920	0	0	0
NoOfOrderedIndexes	128	1	1	1
NoOfTables	128	1	1	1
IndexMemory (KB)	18432	0	0	0
NoOfUniqueHashIndexes	64	0	0	0
NoOfAttributes	1000	3	3	3
NoOfTriggers	768	5	5	5

For debugging purposes, the Perl arrays containing the queries run by this script can be read from the file specified using `--savequeries`; a file containing such arrays to be read in during script execution can be specified using `--loadqueries`. Neither of these options has a default value.

The following information applies to all MySQL Cluster users.

16.9.16. `ndb_waiter` — Wait for Cluster to Reach a Given Status

`ndb_waiter` repeatedly (each 100 milliseconds) prints out the status of all cluster data nodes until either the cluster reaches a given status or the `--timeout` limit is exceeded, then exits. By default, it waits for the cluster to achieve `STARTED` status, in which all nodes have started and connected to the cluster. This can be overridden using the `--no-contact` and `--not-started` options (see [Additional Options](#)).

The node states reported by this utility are as follows:

- **NO_CONTACT**: The node cannot be contacted.
- **UNKNOWN**: The node can be contacted, but its status is not yet known. Usually, this means that the node has received a `START` or `RESTART` command from the management server, but has not yet acted on it.
- **NOT_STARTED**: The node has stopped, but remains in contact with the cluster. This is seen when restarting the node using the management client's `RESTART` command.
- **STARTING**: The node's `ndbd` process has started, but the node has not yet joined the cluster.
- **STARTED**: The node is operational, and has joined the cluster.
- **SHUTTING_DOWN**: The node is shutting down.
- **SINGLE USER MODE**: This is shown for all cluster data nodes when the cluster is in single user mode.

Usage:

```
ndb_waiter [-c connect_string]
```

Additional Options:

- `--no-contact, -n`

Instead of waiting for the `STARTED` state, `ndb_waiter` continues running until the cluster reaches `NO_CONTACT` status before exiting.

- `--not-started`

Instead of waiting for the `STARTED` state, `ndb_waiter` continues running until the cluster reaches `NOT_STARTED` status before exiting.

- `--timeout=seconds, -t seconds`

Time to wait. The program exits if the desired state is not achieved within this number of seconds. The default is 120 seconds (1200 reporting cycles).

Sample Output. Shown here is the output from `ndb_waiter` when run against a 4-node cluster in which two nodes have been shut down and then started again manually. Duplicate reports (indicated by “...”) are omitted.

```
shell> ./ndb_waiter -c localhost
Connecting to mgmsrv at (localhost)
State node 1 STARTED
State node 2 NO_CONTACT
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED
...
State node 1 STARTED
State node 2 UNKNOWN
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED
...
State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED
...
State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 UNKNOWN
Waiting for cluster enter state STARTED
...
State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED
...
State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED
...
State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTED
Waiting for cluster enter state STARTED
NDBT_ProgramExit: 0 - OK
```

Note

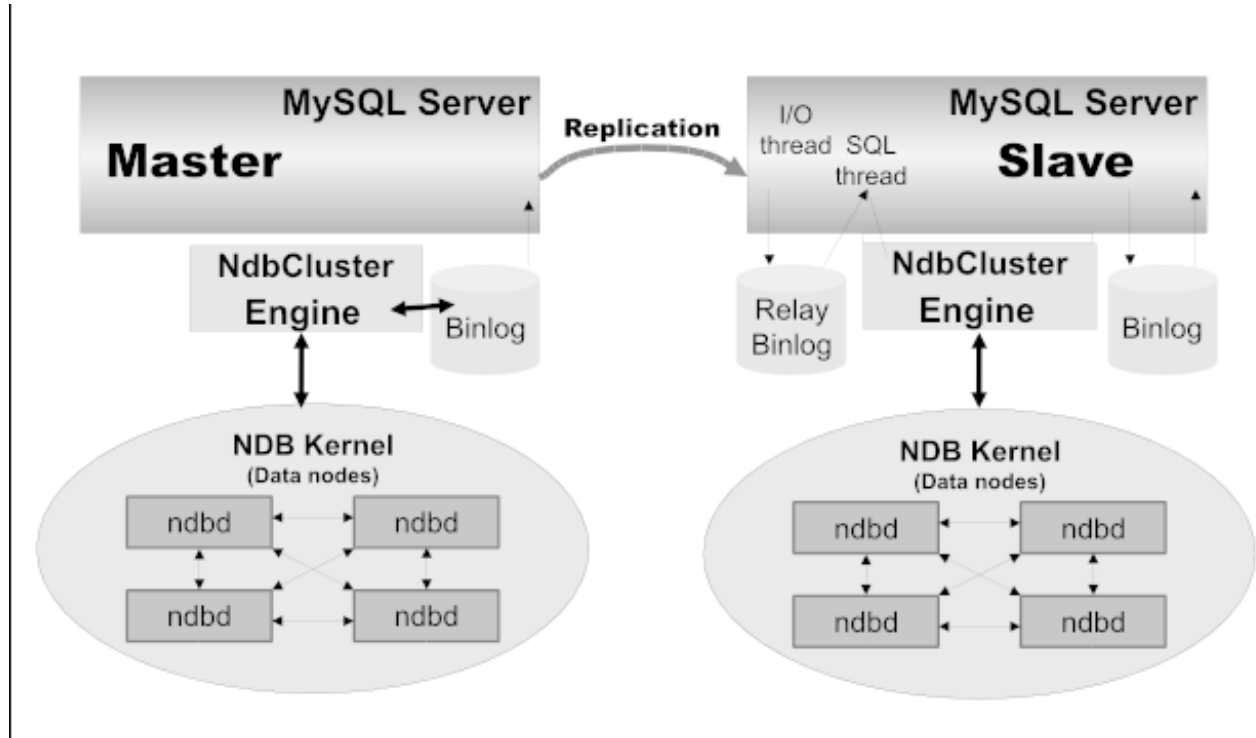
If no connectstring is specified, then `ndb_waiter` tries to connect to a management on `localhost`, and reports `Connecting to mgmsrv at (null)`.

16.10. MySQL Cluster Replication

Previous to MySQL 5.1.6, *asynchronous replication*, more usually referred to simply as “replication”, was not available when using

MySQL Cluster. MySQL 5.1.6 introduces master-slave replication of this type for MySQL Cluster databases. This section explains how to set up and manage a configuration wherein one group of computers operating as a MySQL Cluster replicates to a second computer or group of computers. We assume some familiarity on the part of the reader with standard MySQL replication as discussed elsewhere in this Manual. (See [Chapter 15, Replication](#)).

Normal (non-clustered) replication involves a “master” server and a “slave” server, the master being the source of the operations and data to be replicated and the slave being the recipient of these. In MySQL Cluster, replication is conceptually very similar but can be more complex in practice, as it may be extended to cover a number of different configurations including replicating between two complete clusters. Although a MySQL Cluster itself depends on the [NDB Cluster](#) storage engine for clustering functionality, it is not necessary to use the Cluster storage engine on the slave. However, for maximum availability, it is possible to replicate from one MySQL Cluster to another, and it is this type of configuration that we discuss, as shown in the following figure:



In this scenario, the replication process is one in which successive states of a master cluster are logged and saved to a slave cluster. This process is accomplished by a special thread known as the NDB binlog injector thread, which runs on each MySQL server and produces a binary log ([binlog](#)). This thread ensures that all changes in the cluster producing the binary log — and not just those changes that are effected via the MySQL Server — are inserted into the binary log with the correct serialization order. We refer to the MySQL replication master and replication slave servers as replication servers or replication nodes, and the data flow or line of communication between them as a *replication channel*.

16.10.1. Abbreviations and Symbols

Throughout this section, we use the following abbreviations or symbols for referring to the master and slave clusters, and to processes and commands run on the clusters or cluster nodes:

Symbol or Abbreviation	Description (Refers to...)
<i>M</i>	The cluster serving as the (primary) replication master
<i>S</i>	The cluster acting as the (primary) replication slave
<code>shellM></code>	Shell command to be issued on the master cluster
<code>mysqlM></code>	MySQL client command issued on a single MySQL server running as an SQL node on the master cluster
<code>mysqlM*></code>	MySQL client command to be issued on all SQL nodes participating in the replication master cluster
<code>shellS></code>	Shell command to be issued on the slave cluster
<code>mysqlS></code>	MySQL client command issued on a single MySQL server running as an SQL node on the slave cluster

<code>mysqls*></code>	MySQL client command to be issued on all SQL nodes participating in the replication slave cluster
<code>C</code>	Primary replication channel
<code>C'</code>	Secondary replication channel
<code>M'</code>	Secondary replication master
<code>S'</code>	Secondary replication slave

16.10.2. Assumptions and General Requirements

A replication channel requires two MySQL servers acting as replication servers (one each for the master and slave). For example, this means that in the case of a replication setup with two replication channels (to provide an extra channel for redundancy), there will be a total of four replication nodes, two per cluster.

Replication of a MySQL Cluster as described in this section and those following is dependent on row-based replication. This means that the replication master MySQL server must be started with `--binlog-format=row`, as described in [Section 16.10.6, “Starting Replication \(Single Replication Channel\)”](#). For general information about row-based replication, see [Section 15.1.2, “Replication Formats”](#).

(It is possible to replicate a MySQL Cluster using statement-based replication. However, in this case, the following restrictions apply: All updates to data rows on the cluster acting as the master must be directed to a single MySQL server; It is not possible to replicate a cluster using several MySQL replication processes at the same time; Only changes made at the SQL level are replicated.)

Each MySQL server used for replication in either cluster must be uniquely identified among all the MySQL replication servers participating in either cluster (you cannot have replication servers on both the master and slave clusters sharing the same ID). This can be done by starting each SQL node using the `--server-id=id` option, where `id` is a unique integer. Although it is not strictly necessary, we will assume for purposes of this discussion that all MySQL installations are the same version.

In any event, both MySQL servers involved in replication must be compatible with one another with respect to both the version of the replication protocol used and the SQL feature sets which they support; the simplest and easiest way to assure that this is the case is to use the same MySQL version for all servers involved. Note that in many cases it is not possible to replicate to a slave running a version of MySQL with a lower version number than that of the master — see [Section 15.3.2, “Replication Compatibility Between MySQL Versions”](#), for details.

We assume that the slave server or cluster is dedicated to replication of the master, and that no other data is being stored on it.

16.10.3. Known Issues in MySQL Cluster Replication

The following are known problems or issues when using replication with MySQL Cluster in MySQL 5.1:

- **Loss of master-slave connection.** Prior to MySQL 5.1.18, a MySQL Cluster replication slave `mysqlsld` had no way of detecting that the connection from the master had been interrupted (due to, for instance, the master going down or a network failure). For this reason, it was possible for the slave to become inconsistent with the master.

Beginning with MySQL 5.1.18, the master issues a “gap” event when connecting to the cluster. When the slave encounters a gap in the replication log, it stops with an error message. This message is available in the output of `SHOW SLAVE STATUS`, and indicates that the SQL thread has stopped due to an incident registered in the replication stream, and that manual intervention is required. In order to restart the slave, it is necessary to issue the following commands:

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1;
START SLAVE;
```

The slave then resumes reading the master binlog from the point where the gap was recorded.

(If high availability is a requirement for the slave server or cluster, then it is still advisable to set up multiple replication lines, to monitor the master `mysqlsld` on the primary replication line, and to fail over to a secondary line if and as necessary. For information about implementing this type of setup, see [Section 16.10.7, “Using Two Replication Channels”](#), and [Section 16.10.8, “Implementing Failover with MySQL Cluster”](#).)

- **Multi-byte character sets.** There are several known issues with regard to the use of multi-byte characters sets with MySQL Cluster Replication. See [Bug#27404](#) (fixed in MySQL 5.1.21), [Bug#29562](#), [Bug#29563](#), and [Bug#29564](#) for more information.
- **Circular replication.** Prior to MySQL 5.1.18, circular replication was not supported with MySQL Cluster replication, due to the fact that all log events created in a particular MySQL Cluster were wrongly tagged with the server ID of the MySQL server used as

master and not with the server ID of the originating server.

MySQL Cluster 5.1 Carrier Grade Edition

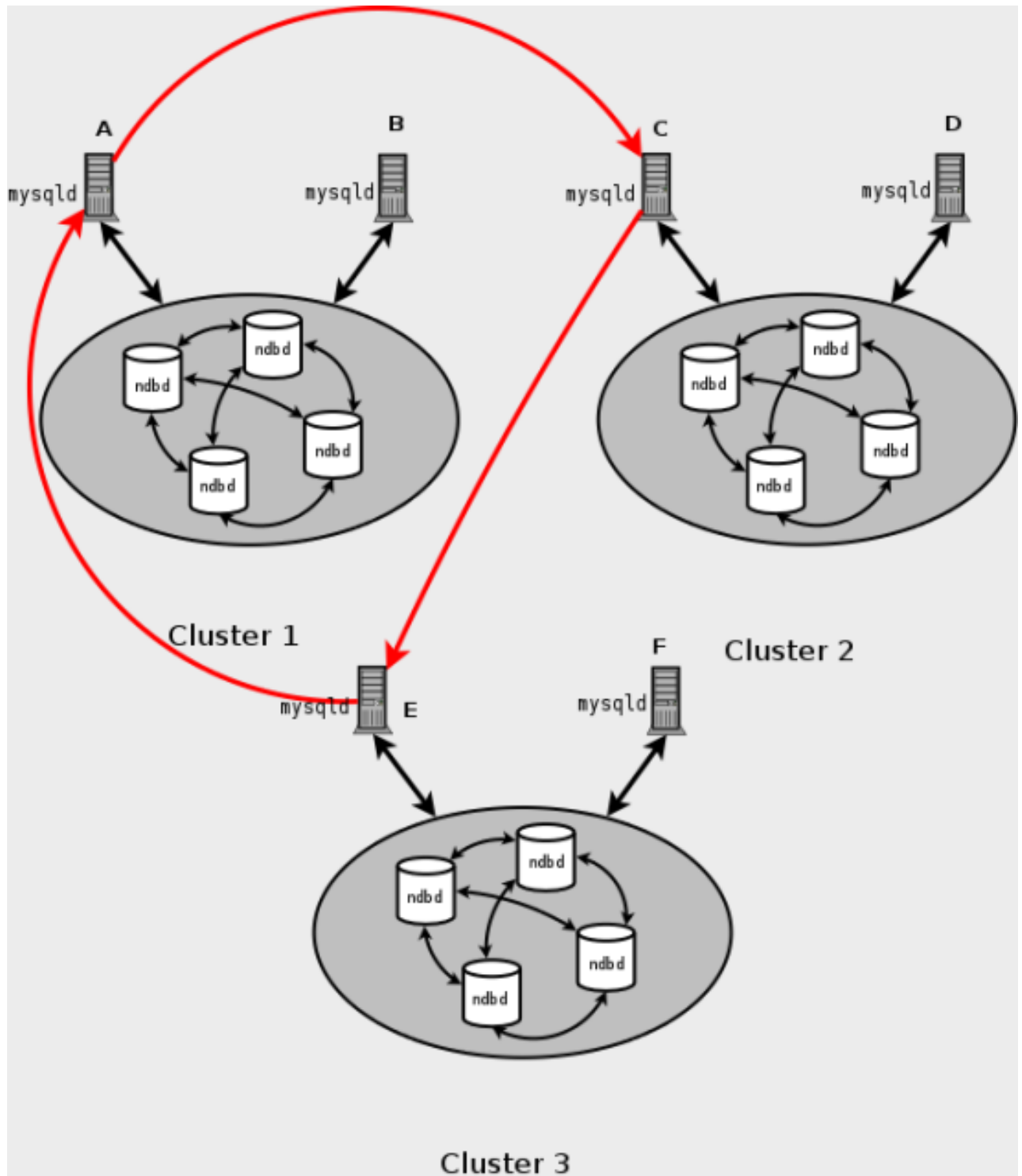
The following information does not apply to users of MySQL Cluster 5.1 Carrier Grade Edition. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

Beginning with MySQL 5.1.18, this limitation is lifted, as discussed in the next few paragraphs, in which we consider the example of a replication setup involving three MySQL Clusters numbered 1, 2, and 3, in which Cluster 1 acts as the replication master for Cluster 2, Cluster 2 acts as the master for Cluster 3, and Cluster 3 acts as the master for Cluster 1. Each cluster has two SQL nodes, with SQL nodes A and B belonging to Cluster 1, SQL nodes C and D belonging to Cluster 2, and SQL nodes E and F belonging to Cluster 3.

Circular replication using these clusters is supported as long as:

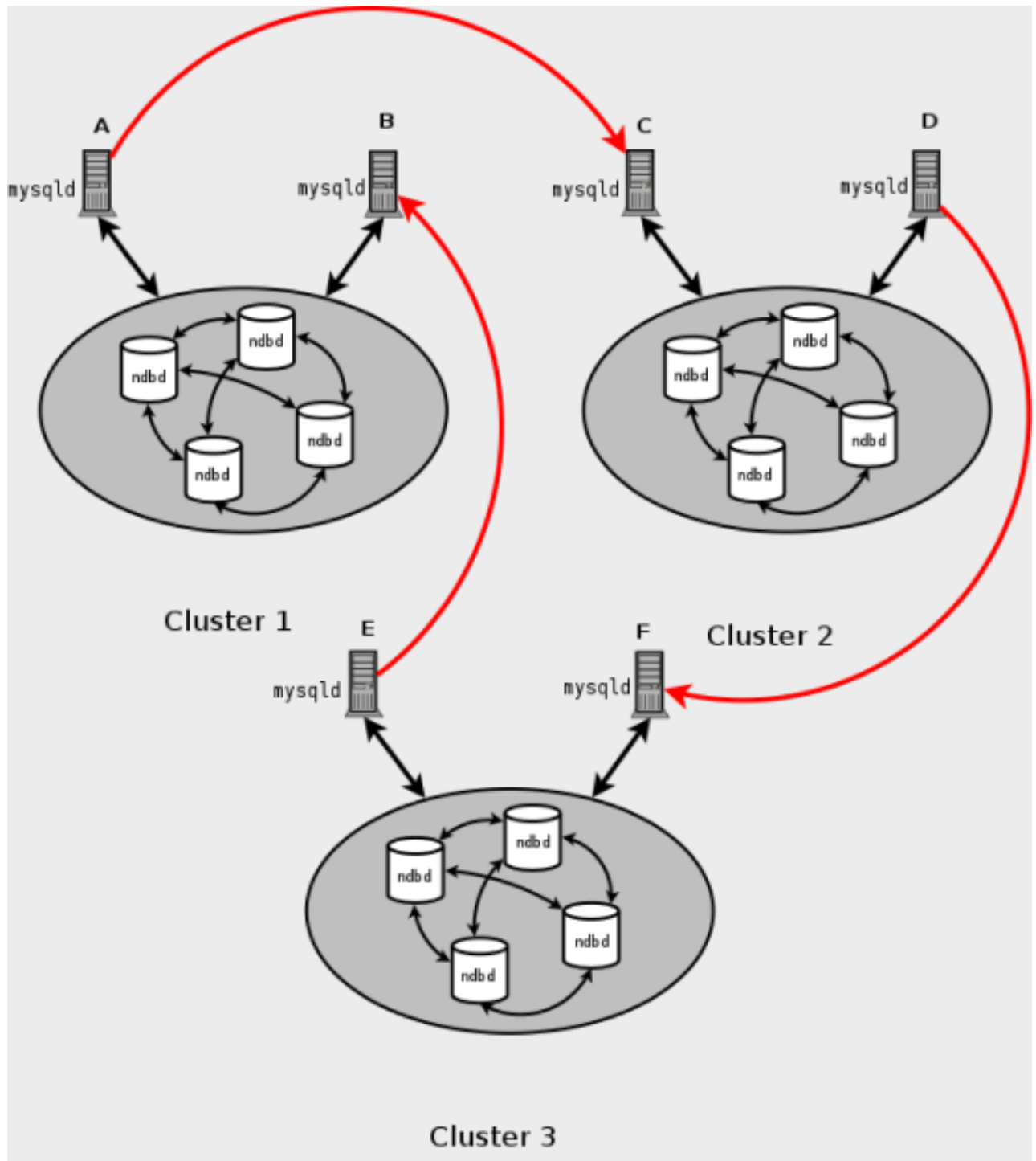
- the SQL nodes on all masters and slaves are the same
- All SQL nodes acting as replication masters and slaves are started using the `--log-slave-updates` option

This type of circular replication setup is shown in the following diagram:



In this scenario, SQL node A in Cluster 1 replicates to SQL node C in Cluster 2; SQL node C replicates to SQL node E in Cluster 3; SQL node E replicates to SQL node A. In other words, the replication line (indicated by the red arrows in the diagram) directly connects all SQL nodes used as replication masters and slaves.

It should also be possible to set up circular replication in which not all master SQL nodes are also slaves, as shown here:



In this case, different SQL nodes in each cluster are used as replication masters and slaves. However, you must *not* start any of the SQL nodes using `--log-slave-updates` (see the [description of this option](#) for more information). This type of circular replication scheme for MySQL Cluster, in which the line of replication (again indicated by the red arrows in the diagram) is discontinuous, should be possible, but it should be noted that it has not yet been thoroughly tested and must therefore still be considered experimental.

The following information applies to all MySQL Cluster users.

- **DDL statements.** The use of data definition statements, such as `CREATE TABLE`, `DROP TABLE`, and `ALTER TABLE`, are recor-

ded in the binary log for only the MySQL server on which they are issued.

- **Explicit primary key required.** In MySQL 5.1.6, only those **NDB** tables having explicit primary keys could be replicated. This limitation was lifted in MySQL 5.1.7.
- **Restarting with `--initial`.** Restarting the cluster with the `--initial` option causes the sequence of GCI and epoch numbers to start over from 0. (This is generally true of MySQL Cluster and not limited to replication scenarios involving Cluster.) The MySQL servers involved in replication should in this case be restarted. After this, you should use the **RESET MASTER** and **RESET SLAVE** statements to clear the invalid `ndb_binlog_index` and `ndb_apply_status` tables, respectively.
- **`auto_increment_offset` and `auto_increment_increment` variables.** The use of the `auto_increment_offset` and `auto_increment_increment` server system variables is supported beginning with MySQL 5.1.20. Previously, these produced unpredictable results when used with **NDB** tables or MySQL Cluster replication.
- **Replication from **NDB** to other storage engines.** If you attempt to replicate from a MySQL Cluster to a slave that uses a storage engine that does not handle its own binary logging, the replication process aborts with the error **BINARY LOGGING NOT POSSIBLE ... STATEMENT CANNOT BE WRITTEN ATOMICALLY SINCE MORE THAN ONE ENGINE INVOLVED AND AT LEAST ONE ENGINE IS SELF-LOGGING** (Error 1595). It is possible to work around this issue in one of the following ways:
 - **Turn off binary logging on the slave.** This can be accomplished by setting `SQL_BIN_LOG = 0`.
 - **Change the storage engine used for the `mysql.ndb_apply_status` table.** Causing this table to use an engine that does not handle its own binary logging can also eliminate the conflict. This can be done by issuing a statement such as **ALTER TABLE `mysql.ndb_apply_status` ENGINE=MyISAM** on the slave. It is safe to do this when using a non-**NDB** storage engine on the slave, since you do not then need to worry about keeping multiple slave SQL nodes synchronized.
 - **Filter out changes to the `mysql.ndb_apply_status` table on the slave.** This can be done by starting the slave SQL node with the option `--replicate-ignore-table=mysql.ndb_apply_status`. If you need for other tables to be ignored by replication, you might wish to use an appropriate `--replicate-wild-ignore-table` option instead.

16.10.4. Cluster Replication Schema and Tables

Replication in MySQL Cluster makes use of a number of dedicated tables in the `mysql` database on each MySQL Server instance acting as an SQL node in both the cluster being replicated and the replication slave (whether the slave is a single server or a cluster). These tables are created during the MySQL installation process by the `mysql_install_db` script, and include a table for storing the binary log's indexing data. Since the `ndb_binlog_index` table is local to each MySQL server and does not participate in clustering, it uses the **MyISAM** storage engine. This means that it must be created separately on each `mysqld` participating in the master cluster. (However, the binlog itself contains updates from all MySQL servers in the cluster to be replicated.) This table is defined as follows:

```
CREATE TABLE `ndb_binlog_index` (
  `Position` BIGINT(20) UNSIGNED NOT NULL,
  `File` VARCHAR(255) NOT NULL,
  `epoch` BIGINT(20) UNSIGNED NOT NULL,
  `inserts` BIGINT(20) UNSIGNED NOT NULL,
  `updates` BIGINT(20) UNSIGNED NOT NULL,
  `deletes` BIGINT(20) UNSIGNED NOT NULL,
  `schemaops` BIGINT(20) UNSIGNED NOT NULL,
  PRIMARY KEY (`epoch`)
) ENGINE=MYISAM DEFAULT CHARSET=latin1;
```

Important

Prior to MySQL 5.1.14, the `ndb_binlog_index` table was known as `binlog_index`, and was kept in a separate `cluster` database, which in MySQL 5.1.7 and earlier was known as the `cluster_replication` database. Similarly, the `ndb_apply_status` and `ndb_schema` tables were known as `apply_status` and `schema`, and were also found in the `cluster` (earlier `cluster_replication`) database. However, beginning with MySQL 5.1.14, all MySQL Cluster replication tables reside in the `mysql` system database.

Information about how this change affects upgrades from MySQL Cluster 5.1.13 and earlier to 5.1.14 and later versions can be found in [Section C.1.15, “Changes in release 5.1.14 \(05 December 2006\)”](#).

The following figure shows the relationship of the MySQL Cluster replication master server, its binlog injector thread, and the `mysql.ndb_binlog_index` table.

[REDACTED]

However, it is advisable to check for the existence and integrity of these tables as an initial step in preparing a MySQL Cluster for replication. It is possible to view event data recorded in the binary log by querying the `mysql.binlog_index` table directly on the master. This can be also be accomplished using the `SHOW BINLOG EVENTS` statement on either the replication master or slave MySQL servers. (See [Section 12.6.1.4, “SHOW BINLOG EVENTS Syntax”](#).)

You can also obtain useful information from the output of `SHOW ENGINE NDB STATUS`.

Note

Beginning with MySQL 5.1.14, if the `apply_status` table does not exist on the slave, it is created by `ndb_restore`. ([Bug#14612](#))

16.10.5. Preparing the Cluster for Replication

Preparing the MySQL Cluster for replication consists of the following steps:

1. Check all MySQL servers for version compatibility (see [Section 16.10.2, “Assumptions and General Requirements”](#)).
2. Create a slave account on the master Cluster with the appropriate privileges:

```
mysqlM> GRANT REPLICATION SLAVE
-> ON *.* TO 'slave_user'@'slave_host'
-> IDENTIFIED BY 'slave_password';
```

In the previous statement, `slave_user` is the slave account username, `slave_host` is the hostname or IP address of the replication slave, and `slave_password` is the password to assign to this account.

For example, to create a slave user account with the name “`myslave`,” logging in from the host named “`rep-slave`,” and using the password “`53cr37`,” use the following `GRANT` statement:

```
mysqlM> GRANT REPLICATION SLAVE
-> ON *.* TO 'myslave'@'rep-slave'
-> IDENTIFIED BY '53cr37';
```

For security reasons, it is preferable to use a unique user account — not employed for any other purpose — for the replication slave account.

3. Configure the slave to use the master. Using the MySQL Monitor, this can be accomplished with the `CHANGE MASTER TO` statement:

```
mysqlS> CHANGE MASTER TO
-> MASTER_HOST='master_host',
-> MASTER_PORT=master_port,
-> MASTER_USER='slave_user',
-> MASTER_PASSWORD='slave_password';
```

In the previous statement, `master_host` is the hostname or IP address of the replication master, `master_port` is the port for the slave to use for connecting to the master, `slave_user` is the username set up for the slave on the master, and `slave_password` is the password set for that user account in the previous step.

For example, to tell the slave to replicate from the MySQL server whose hostname is “`rep-master`,” using the replication slave account created in the previous step, use the following statement:

```
mysqlS> CHANGE MASTER TO
-> MASTER_HOST='rep-master'
-> MASTER_PORT=3306,
-> MASTER_USER='myslave'
-> MASTER_PASSWORD='53cr37';
```

(For a complete list of clauses that can be used with this statement, see [Section 12.6.2.1, “CHANGE MASTER TO Syntax”](#).)

You can also configure the slave to use the master by setting the corresponding startup options in the slave server's `my.cnf` file. To configure the slave in the same way as the preceding example `CHANGE MASTER TO` statement, the following information would need to be included in the slave's `my.cnf` file:

```
[mysqld]
```

```
master-host=rep-master
master-port=3306
master-user=myslave
master-password=53cr37
```

See [Section 15.1.3, “Replication Options and Variables”](#), for additional options that can be set in `my.cnf` for replication slaves.

Note

To provide replication backup capability, you will also need to add an `ndb-connectstring` option to the slave's `my.cnf` file prior to starting the replication process. See [Section 16.10.9, “MySQL Cluster Backups With Replication”](#), for details.

4. If the master cluster is already in use, you can create a backup of the master and load this onto the slave to cut down on the amount of time required for the slave to synchronize itself with the master. If the slave is also running MySQL Cluster, this can be accomplished using the backup and restore procedure described in [Section 16.10.9, “MySQL Cluster Backups With Replication”](#).

```
ndb-connectstring=management_host[:port]
```

In the event that you are *not* using MySQL Cluster on the replication slave, you can create a backup with this command on the replication master:

```
shellM> mysqldump --master-data=1
```

Then import the resulting data dump onto the slave by copying the dump file over to the slave. After this, you can use the `mysql` client to import the data from the dumpfile into the slave database as shown here, where `dump_file` is the name of the file that was generated using `mysqldump` on the master, and `db_name` is the name of the database to be replicated:

```
shellS> mysql -u root -p db_name < dump_file
```

For a complete list of options to use with `mysqldump`, see [Section 7.13, “mysqldump — A Database Backup Program”](#).

Note

If you copy the data to the slave in this fashion, you should make sure that the slave is started with the `-skip-slave-start` option on the command line, or else include `skip-slave-start` in the slave's `my.cnf` file to keep it from trying to connect to the master to begin replicating before all the data has been loaded. Once the data load-in has completed, follow the additional steps outlined in the next two sections.

5. Ensure that each MySQL server acting as a replication master is configured with a unique server ID, and with binary logging enabled, using the row format. (See [Section 15.1.2, “Replication Formats”](#).) These options can be set either in the master server's `my.cnf` file, or on the command line when starting the master `mysqld` process. See [Section 16.10.6, “Starting Replication \(Single Replication Channel\)”](#), for information regarding the latter option.

16.10.6. Starting Replication (Single Replication Channel)

This section outlines the procedure for starting MySQL Cluster replication using a single replication channel.

1. Start the MySQL replication master server by issuing this command:

```
shellM> mysqld --ndbcluster --server-id=id \
--log-bin --binlog-format=row &
```

In the previous statement, `id` is this server's unique ID (see [Section 16.10.2, “Assumptions and General Requirements”](#)). This starts the server's `mysqld` process with binary logging enabled using the proper logging format.

2. Start the MySQL replication slave server as shown here:

```
shellS> mysqld --ndbcluster --server-id=id &
```

In the previous statement, `id` is the slave server's unique ID. It is not necessary to enable logging on the replication slave.

Note

You should use the `--skip-slave-start` option with this command or else you should include `skip-slave-start` in the slave server's `my.cnf` file, unless you want replication to begin immediately. With the use of this option, the start of replication is delayed until the appropriate `START SLAVE` statement has been issued, as explained in Step 4 below.

- It is necessary to synchronize the slave server with the master server's replication binlog. If binary logging has not previously been running on the master, run the following statement on the slave:

```
mysql> CHANGE MASTER TO
-> MASTER_LOG_FILE=' ',
-> MASTER_LOG_POS=4;
```

This instructs the slave to begin reading the master's binary log from the log's starting point. Otherwise — that is, if you are loading data from the master using a backup — see [Section 16.10.8, “Implementing Failover with MySQL Cluster”](#), for information on how to obtain the correct values to use for `MASTER_LOG_FILE` and `MASTER_LOG_POS` in such cases.

- Finally, you must instruct the slave to begin applying replication by issuing this command from the `mysql` client on the replication slave:

```
mysql> START SLAVE;
```

This also initiates the transmission of replication data from the master to the slave.

It is also possible to use two replication channels, in a manner similar to the procedure described in the next section; the differences between this and using a single replication channel are covered in [Section 16.10.7, “Using Two Replication Channels”](#).

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see [Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”](#).

Beginning with MySQL 5.1.19-ndb-6.2.3, it is possible to improve cluster replication performance by enabling *batched updates*. This can be accomplished by starting slave `mysqld` processes with the `--slave-allow-batching` option. Normally, updates are applied as soon as they are received. However, the use of batching causes updates to be applied in 32 KB batches, which can result in higher throughput and less CPU usage, particularly where individual updates are relatively small.

Note

Slave batching works on a per-epoch basis; updates belonging to more than one transaction can be sent as part of the same batch.

All outstanding updates are applied when the end of an epoch is reached, even if the updates total less than 32 KB.

Batching can be turned on and off at runtime. To activate it at runtime, you can use either of these two statements:

```
SET GLOBAL slave_allow_batching = 1;
SET GLOBAL slave_allow_batching = ON;
```

If a particular batch causes problems (such as a statement whose effects do not appear to be replicated correctly), slave batching can be deactivated using either of the following statements:

```
SET GLOBAL slave_allow_batching = 0;
SET GLOBAL slave_allow_batching = OFF;
```

You can check whether slave batching is currently being used by means of an appropriate `SHOW VARIABLES` statement, like this one:

```
mysql> SHOW VARIABLES LIKE 'slave%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| slave_allow_batching | ON    |
| slave_compressed_protocol | OFF   |
| slave_load_tmpdir | /tmp  |
| slave_net_timeout | 3600  |
| slave_skip_errors | OFF   |
+-----+-----+
```

```
| slave_transaction_retries | 10 |
+-----+-----+
6 rows in set (0.00 sec)
```

The following information applies to all MySQL Cluster users.

16.10.7. Using Two Replication Channels

In a more complete example scenario, we envision two replication channels to provide redundancy and thereby guard against possible failure of a single replication channel. This requires a total of four replication servers, two masters for the master cluster and two slave servers for the slave cluster. For purposes of the discussion that follows, we assume that unique identifiers are assigned as shown here:

Server ID	Description
1	Master - primary replication channel (<i>M</i>)
2	Master - secondary replication channel (<i>M'</i>)
3	Slave - primary replication channel (<i>S</i>)
4	Slave - secondary replication channel (<i>S'</i>)

Setting up replication with two channels is not radically different from setting up a single replication channel. First, the `mysqld` processes for the primary and secondary replication masters must be started, followed by those for the primary and secondary slaves. Then the replication processes may be initiated by issuing the `START SLAVE` statement on each of the slaves. The commands and the order in which they need to be issued are shown here:

1. Start the primary replication master:

```
shellM> mysqld --ndbcluster --server-id=1 \
              --log-bin --binlog-format=row &
```

2. Start the secondary replication master:

```
shellM'> mysqld --ndbcluster --server-id=2 \
              --log-bin --binlog-format=row &
```

3. Start the primary replication slave server:

```
shellS> mysqld --ndbcluster --server-id=3 \
              --skip-slave-start &
```

4. Start the secondary replication slave:

```
shellS'> mysqld --ndbcluster --server-id=4 \
              --skip-slave-start &
```

5. Finally, initiate replication on the primary channel by executing the `START SLAVE` statement on the primary slave as shown here:

```
mysqlS> START SLAVE;
```

Warning

Only the primary channel is to be started at this point. The secondary replication channel is to be started only in the event that the primary replication channel fails, as described in [Section 16.10.8, “Implementing Failover with MySQL Cluster”](#). Running multiple replication channels simultaneously can result in unwanted duplicate records being created on the replication slaves.

As mentioned previously, it is not necessary to enable binary logging on replication slaves.

16.10.8. Implementing Failover with MySQL Cluster

In the event that the primary Cluster replication process fails, it is possible to switch over to the secondary replication channel. The fol-

lowing procedure describes the steps required to accomplish this.

1. Obtain the time of the most recent global checkpoint (GCP). That is, you need to determine the most recent epoch from the `ndb_apply_status` table on the slave cluster, which can be found using the following query:

```
mysqlS'> SELECT @latest:=MAX(epoch)
-> FROM mysql.ndb_apply_status;
```

2. Using the information obtained from the query shown in Step 1, obtain the corresponding records from the `ndb_binlog_index` table on the master cluster as shown here:

```
mysqlM'> SELECT
-> @file:=SUBSTRING_INDEX(File, '/', -1),
-> @pos:=Position
-> FROM mysql.ndb_binlog_index
-> WHERE epoch > @latest
-> ORDER BY epoch ASC LIMIT 1;
```

These are the records saved on the master since the failure of the primary replication channel. We have employed a user variable `@latest` here to represent the value obtained in Step 1. Of course, it is not possible for one `mysqld` instance to access user variables set on another server instance directly. These values must be “plugged in” to the second query manually or in application code.

3. Now it is possible to synchronize the secondary channel by running the following query on the secondary slave server:

```
mysqlS'> CHANGE MASTER TO
-> MASTER_LOG_FILE='@file',
-> MASTER_LOG_POS=@pos;
```

Again we have employed user variables (in this case `@file` and `@pos`) to represent the values obtained in Step 2 and applied in Step 3; in practice these values must be inserted manually or using application code that can access both of the servers involved.

Note

`@file` is a string value such as `'/var/log/mysql/replication-master-bin.00001'`, and so must be quoted when used in SQL or application code. However, the value represented by `@pos` must *not* be quoted. Although MySQL normally attempts to convert strings to numbers, this case is an exception.

4. You can now initiate replication on the secondary channel by issuing the appropriate command on the secondary slave `mysqld`:

```
mysqlS'> START SLAVE;
```

Once the secondary replication channel is active, you can investigate the failure of the primary and effect repairs. The precise actions required to do this will depend upon the reasons for which the primary channel failed.

Warning

The secondary replication channel is to be started only if and when the primary replication channel has failed. Running multiple replication channels simultaneously can result in unwanted duplicate records being created on the replication slaves.

If the failure is limited to a single server, it should (in theory) be possible to replicate from *M* to *S'*, or from *M'* to *S*; however, this has not yet been tested.

16.10.9. MySQL Cluster Backups With Replication

This section discusses making backups and restoring from them using MySQL Cluster replication. We assume that the replication servers have already been configured as covered previously (see [Section 16.10.5, “Preparing the Cluster for Replication”](#), and the sections immediately following). This having been done, the procedure for making a backup and then restoring from it is as follows:

1. There are two different methods by which the backup may be started.

- **Method A:**

This method requires that the cluster backup process was previously enabled on the master server, prior to starting the replication process. This can be done by including the line

```
ndb-connectstring=management_host[:port]
```

in a `[MYSQL_CLUSTER]` section in the `my.cnf` file, where *management_host* is the IP address or hostname of the NDB management server for the master cluster, and *port* is the management server's port number. Note that the port number needs to be specified only if the default port (1186) is not being used. (See [Section 16.3.3, “Multi-Computer Configuration”](#), for more information about ports and port allocation in MySQL Cluster.)

In this case, the backup can be started by executing this statement on the replication master:

```
shellM> ndb_mgm -e "START BACKUP"
```

- **Method B:**

If the `my.cnf` file does not specify where to find the management host, you can start the backup process by passing this information to the NDB management client as part of the `START BACKUP` command, like this:

```
shellM> ndb_mgm management_host:port -e "START BACKUP"
```

where *management_host* and *port* are the hostname and port number of the management server. In our scenario as outlined earlier (see [Section 16.10.5, “Preparing the Cluster for Replication”](#)), this would be executed as follows:

```
shellM> ndb_mgm rep-master:1186 -e "START BACKUP"
```

In either case, it is highly advisable to allow any pending transactions to be completed before beginning the backup, and then not to permit any new transactions to begin during the backup process.

2. Copy the cluster backup files to the slave that is being brought on line. Each system running an `ndbd` process for the master cluster will have cluster backup files located on it, and *all* of these files must be copied to the slave to ensure a successful restore. The backup files can be copied into any directory on the computer where the slave management host resides, so long as the MySQL and NDB binaries have read permissions in that directory. In this case, we will assume that these files have been copied into the directory `/var/BACKUPS/BACKUP-1`.

It is not necessary that the slave cluster have the same number of `ndbd` processes (data nodes) as the master; however, it is highly recommended this number be the same. It is necessary that the slave be started with the `--skip-slave-start` option, to prevent premature startup of the replication process.

3. Create any databases on the slave cluster that are present on the master cluster that are to be replicated to the slave.

Important

A `CREATE DATABASE` (or `CREATE SCHEMA`) statement corresponding to each database to be replicated must be executed on each SQL node in the slave cluster.

4. Reset the slave cluster using this statement in the MySQL Monitor:

```
mysqlS> RESET SLAVE;
```

It is important to make sure that the slave's `apply_status` table does not contain any records prior to running the restore process. You can accomplish this by running this SQL statement on the slave:

```
mysqlS> DELETE FROM mysql.ndb_apply_status;
```

5. You can now start the cluster restoration process on the replication slave using the `ndb_restore` command for each backup file in turn. For the first of these, it is necessary to include the `-m` option to restore the cluster metadata:

```
shellS> ndb_restore -c slave_host:port -n node-id \
-b backup-id -m -r dir
```

`dir` is the path to the directory where the backup files have been placed on the replication slave. For the `ndb_restore` commands corresponding to the remaining backup files, the `-m` option should *not* be used.

For restoring from a master cluster with four data nodes (as shown in the figure in [Section 16.10, “MySQL Cluster Replication”](#)) where the backup files have been copied to the directory `/var/BACKUPS/BACKUP-1`, the proper sequence of commands to be executed on the slave might look like this:

```
shellS> ndb_restore -c rep-slave:1186 -n 2 -b 1 -m \
-r ./var/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 3 -b 1 \
-r ./var/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 4 -b 1 \
-r ./var/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 5 -b 1 -e \
-r ./var/BACKUPS/BACKUP-1
```

Important

The `-e` (or `--restore-epoch`) option in the final invocation of `ndb_restore` in this example is required in order that the epoch is written to the slave `mysql.ndb_apply_status`. Without this information, the slave will not be able to synchronize properly with the master. (See [Section 16.8.3, “ndb_restore — Restore a Cluster Backup”](#).)

- Now you need to obtain the most recent epoch from the `ndb_apply_status` table on the slave (as discussed in [Section 16.10.8, “Implementing Failover with MySQL Cluster”](#)):

```
mysqlS> SELECT @latest:=MAX(epoch)
FROM mysql.ndb_apply_status;
```

- Using `@latest` as the epoch value obtained in the previous step, you can obtain the correct starting position `@pos` in the correct binary log file `@file` from the master's `mysql.ndb_binlog_index` table using the query shown here:

```
mysqlM> SELECT
-> @file:=SUBSTRING_INDEX(File, '/', -1),
-> @pos:=Position
-> FROM mysql.ndb_binlog_index
-> WHERE epoch > @latest
-> ORDER BY epoch ASC LIMIT 1;
```

- Using the values obtained in the previous step, you can now issue the appropriate `CHANGE MASTER TO` statement in the slave's `mysql` client:

```
mysqlS> CHANGE MASTER TO
-> MASTER_LOG_FILE='@file',
-> MASTER_LOG_POS=@pos;
```

- Now that the slave “knows” from what point in which `binlog` file to start reading data from the master, you can cause the slave to begin replicating with this standard MySQL statement:

```
mysqlS> START SLAVE;
```

To perform a backup and restore on a second replication channel, it is necessary only to repeat these steps, substituting the hostnames and IDs of the secondary master and slave for those of the primary master and slave replication servers where appropriate, and running the preceding statements on them.

For additional information on performing Cluster backups and restoring Cluster from backups, see [Section 16.8, “On-line Backup of MySQL Cluster”](#).

16.10.9.1. Automating Synchronization of the Slave to the Master binlog

It is possible to automate much of the process described in the previous section (see [Section 16.10.9, “MySQL Cluster Backups With Replication”](#)). The following Perl script `reset-slave.pl` serves as an example of how you can do this.

```
#!/user/bin/perl -w
# file: reset-slave.pl
# Copyright ©2005 MySQL AB
```

```

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to:
# Free Software Foundation, Inc.
# 59 Temple Place, Suite 330
# Boston, MA 02111-1307 USA
#
# Version 1.1

##### Includes #####

use DBI;

##### Globals #####

my $m_host='';
my $m_port='';
my $m_user='';
my $m_pass='';
my $s_host='';
my $s_port='';
my $s_user='';
my $s_pass='';
my $dbhM='';
my $dbhS='';

##### Sub Prototypes #####

sub CollectCommandPromptInfo;
sub ConnectToDatabases;
sub DisconnectFromDatabases;
sub GetSlaveEpoch;
sub GetMasterInfo;
sub UpdateSlave;

##### Program Main #####

CollectCommandPromptInfo;
ConnectToDatabases;
GetSlaveEpoch;
GetMasterInfo;
UpdateSlave;
DisconnectFromDatabases;

##### Collect Command Prompt Info #####

sub CollectCommandPromptInfo
{
    ### Check that user has supplied correct number of command line args
    die "Usage:\n
        reset-slave MySQL host< >master MySQL port< \n
                >master user< >master pass< >slave MySQL host< \n
                >slave MySQL port< >slave user< >slave pass< \n
        All 8 arguments must be passed. Use BLANK for NULL passwords\n"
        unless @ARGV == 8;

    $m_host = $ARGV[0];
    $m_port = $ARGV[1];
    $m_user = $ARGV[2];
    $m_pass = $ARGV[3];
    $s_host = $ARGV[4];
    $s_port = $ARGV[5];
    $s_user = $ARGV[6];
    $s_pass = $ARGV[7];

    if ($m_pass eq "BLANK") { $m_pass = ''; }
    if ($s_pass eq "BLANK") { $s_pass = ''; }
}

##### Make connections to both databases #####

sub ConnectToDatabases
{
    ### Connect to both master and slave cluster databases

    ### Connect to master
    $dbhM
    = DBI->connect(
        "dbi:mysql:database=mysql;host=$m_host;port=$m_port",

```



```

    "$m_user" "$m_pass")
    or die "Can't connect to Master Cluster MySQL process!
    Error: $DBI::errstr\n";

### Connect to slave
$dbhS
= DBI->connect(
    "dbi:mysql:database=mysql;host=$s_host",
    "$s_user", "$s_pass")
    or die "Can't connect to Slave Cluster MySQL process!
    Error: $DBI::errstr\n";
}

##### Disconnect from both databases #####
sub DisconnectFromDatabases
{
    ### Disconnect from master

    $dbhM->disconnect
    or warn " Disconnection failed: $DBI::errstr\n";

    ### Disconnect from slave

    $dbhS->disconnect
    or warn " Disconnection failed: $DBI::errstr\n";
}

##### Find the last good GCI #####
sub GetSlaveEpoch
{
    $sth = $dbhS->prepare("SELECT MAX(epoch)
        FROM mysql.ndb_apply_status;")
    or die "Error while preparing to select epoch from slave: ",
        $dbhS->errstr;

    $sth->execute
    or die "Selecting epoch from slave error: ", $sth->errstr;

    $sth->bind_col (1, \ $epoch);
    $sth->fetch;
    print "\tSlave Epoch = $epoch\n";
    $sth->finish;
}

##### Find the position of the last GCI in the binlog #####
sub GetMasterInfo
{
    $sth = $dbhM->prepare("SELECT
        SUBSTRING_INDEX(File, '/', -1), Position
        FROM mysql.ndb_binlog_index
        WHERE epoch > $epoch
        ORDER BY epoch ASC LIMIT 1;")
    or die "Prepare to select from master error: ", $dbhM->errstr;

    $sth->execute
    or die "Selecting from master error: ", $sth->errstr;

    $sth->bind_col (1, \ $binlog);
    $sth->bind_col (2, \ $binpos);
    $sth->fetch;
    print "\tMaster bin log = $binlog\n";
    print "\tMaster Bin Log position = $binpos\n";
    $sth->finish;
}

##### Set the slave to process from that location #####
sub UpdateSlave
{
    $sth = $dbhS->prepare("CHANGE MASTER TO
        MASTER_LOG_FILE='$binlog',
        MASTER_LOG_POS=$binpos;")
    or die "Prepare to CHANGE MASTER error: ", $dbhS->errstr;

    $sth->execute
    or die "CHNAGE MASTER on slave error: ", $sth->errstr;
    $sth->finish;
    print "\tSlave has been updated. You may now start the slave.\n";
}

# end reset-slave.pl

```

16.10.10. MySQL Cluster Replication Conflict Resolution

MySQL Cluster 5.1 Carrier Grade Edition

The information in this section applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

When using a replication setup involving multiple masters, it is possible that different masters may try to update the same row on the slave with different data. Conflict resolution in MySQL Cluster Replication provides a means of resolving such conflicts by allowing a user defined “timestamp” column to be used to determine whether or not an update to the row on a given master should be applied on the slave. Generally speaking, if the “timestamp” for a given row coming from the master is higher than that on the slave, it is applied it; otherwise it is not applied on the slave.

It is available in MySQL Cluster 5.1 Carrier Grade Edition releases beginning with MySQL 5.1.19-ndb-6.3.0.

This replication scheme, also known as “changed parameter only replication”, is configurable on a per-table basis.

- On the master, it must be determined which columns to send (all columns or only those that have been updated).
- On the slave, it must be determined which type of conflict resolution to apply (none or “latest timestamp wins”).

If only some but not all columns are sent, then the master and slave can diverge.

Note

We refer to the column used for determining updates as a “timestamp” column, but the data type of this column is never **TIMESTAMP**; rather, its data type can be either of **INT SIGNED** or **INT UNSIGNED**.

We can see update operations in terms of “before” and “after” images — that is, the states of the table before and after the update is applied. Normally, when updating a table with a primary key, the “before” image is not of great interest; however, when we need to determine on a per-update basis whether or not to use the updated values on a replication slave, we need to make sure that both images are written to the master's binary log. This is done with the **--ndb-log-update-as-write** startup option for **mysqld**, as described later in this section.

Whether logging of complete rows or of updated columns only is done is decided when the MySQL server is started, and cannot be changed online; you must either restart **mysqld**, or start a new **mysqld** instance with different logging options.

For purposes of conflict resolution, there are two basic methods of logging rows:

- Log complete rows
- Log only column data that has been updated — that is, column data whose value has been set, regardless of whether or not this value was actually changed.

Either of the above logging methods can be configured to be done with or without the “before” image.

mysqld startup options. The following **mysqld** startup options are available to control conflict resolution:

- **--ndb-log-update-as-write**. Because conflict resolution is done in the MySQL Server's update handler, it is necessary to control logging on the master such that updates are updates and not writes as in in mainline MySQL 5.1. This option is turned on by default; to turn it off, start the server with **--ndb-log-update-as-write=0** or **--ndb-log-update-as-write=OFF**.
- **--ndb-log-updated-only**. In general it is preferable to log full rows. However, depending on the application, it may be sufficient to log only the updates, and can be more efficient to do so. This can be done by setting **--ndb-log-updated-only** to **1** or **ON**.

To enable conflict resolution, it is necessary to create an **ndb_replication** table in the **mysql** system database on the master, the slave, or both, depending on the conflict resolution type and method to be employed. This table is used in order to control logging and conflict resolution function on a per-table basis, and has one row per table involved in replication. Each row in **mysql.ndb_replication** corresponding to a given table specifies how to log and resolve conflicts for that table. The definition of this table is shown here:

```
CREATE TABLE mysql.ndb_replication (
  db VARBINARY(63),
  table_name VARBINARY(63),
```

```

server_id INT UNSIGNED,
binlog_type INT UNSIGNED,
conflict_fn VARBINARY(128),
PRIMARY KEY USING HASH (db, table_name, server_id)
)
ENGINE=NDB
PARTITION BY KEY(db, table_name);

```

The columns in this table are described in the following list:

- **db**. The name of the database containing the table to be replicated.
- **table_name**. The name of the table to be replicated.
- **server_id**. The unique server ID of the MySQL instance (SQL node) where the table resides.
- **binlog_type**. The type of binary logging to be employed. This is determined as shown in the following table:

Value	Internal Value	Description
0	NBT_DEFAULT	Use server default
1	NBT_NO_LOGGING	Do not log this table in the binary log
2	NBT_UPDATED_ONLY	Only updated attributes are logged
3	NBT_FULL	Log full row, even if not updated (MySQL server default behavior)
4	NBT_USE_UPDATE	(For generating NBT_UPDATED_ONLY_USE_UPDATE and NBT_FULL_USE_UPDATE values only — not intended for separate use)
5	[Not used]	---
6	NBT_UPDATED_ONLY_USE_UPDATE (= NBT_UPDATED_ONLY NBT_USE_UPDATE)	Use updated attributes, even if values are unchanged
7	NBT_FULL_USE_UPDATE(= NBT_FULL NBT_USE_UPDATE)	Use full row, even if values are unchanged

- **conflict_fn**. The conflict resolution function to be applied. Currently this function must be specified as either `NDB$MAX(column_name)` or `NULL` (if conflict resolution is not to be used).

Important

The `mysql.ndb_replication` table is read when the table is set up for replication, so the row corresponding to a table to be replicated must be inserted into `mysql.ndb_replication` *before* creation of the table to be replicated takes place.

Example. Suppose you wish to enable conflict resolution on table `test.t1`, using column `mycol` as the “timestamp”. This can be done using the following two steps:

- On the master, perform this query:

```

INSERT INTO mysql.ndb_replication VALUES
("test", "t1", 0, NULL, "NDB$MAX(mycol)");

```

Inserting a 0 into the `server_id` indicates that all SQL nodes accessing this table should use the conflict resolution. If you want to use conflict resolution on a specific `mysqld` only, use the actual server ID.

Inserting `NULL` into the `binlog_type` column has the same effect as inserting 0 (`NBT_DEFAULT`); the server default is used.

- Create the `test.t1` table:

```

CREATE TABLE test.t1 (
  columns
  mycol INT UNSIGNED,
  columns

```

```
);
```

Now, when updates are done on this table, conflict resolution will be applied, and the version of the row having the greatest value for `mycol` will be written to the slave.

16.11. MySQL Cluster Disk Data Tables

Beginning with MySQL 5.1.6, it is possible to store the non-indexed columns of `NDB` tables on disk, rather than in RAM as with previous versions of MySQL Cluster.

As part of implementing Cluster Disk Data work, a number of improvements were made in MySQL Cluster for the efficient handling of very large amounts (terabytes) of data during node recovery and restart. These include a “no-steal” algorithm for synchronising a starting node with very large data sets. For more information, see the paper [Recovery Principles of MySQL Cluster 5.1](#), by MySQL Cluster developers Mikael Ronström and Jonas Öreland.

16.11.1. Disk Data Objects

This section discusses Disk Data objects — which include tables, log file groups, and tablespaces — as well as how to create and drop them.

Assuming that you have already set up a MySQL Cluster with all nodes (including management and SQL nodes) running MySQL 5.1.6 or newer, the basic steps for creating a Cluster table on disk are as follows:

1. Create a *log file group*, and assign one or more undo log files to it (an undo log file is also referred to as an *undofile*).

Note

In MySQL 5.1, undo log files are necessary only for Disk Data tables. They are no longer used for tables that are stored in memory.

2. Create a *tablespace*, and assign the log file group to it, as well as one or more data files.
3. Create a Disk Data table that uses this tablespace for data storage.

Each of these tasks can be accomplished using SQL statements, as shown in the following example.

1. We create a log file group named `lg_1` using `CREATE LOGFILE GROUP`. This log file group is to be made up of two undo log files, which we name `undo_1.dat` and `undo_2.dat`, whose initial sizes are 16 MB and 12 MB, respectively. (The default initial size for an undo log file is 128 MB.) Optionally, you can also specify a size for the log file group's `UNDO` buffer, or allow it to assume the default value of 8 MB. In this example, we set the `UNDO` buffer's size at 2 MB. A log file group must be created with an undo log file; so we add `undo_1.dat` to `lg_1` in this `CREATE LOGFILE GROUP` statement:

```
CREATE LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo_1.dat'
  INITIAL_SIZE 16M
  UNDO_BUFFER_SIZE 2M
  ENGINE NDB;
```

To add `undo_2.dat` to the log file group, use the following `ALTER LOGFILE GROUP` statement:

```
ALTER LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo_2.dat'
  INITIAL_SIZE 12M
  ENGINE NDB;
```

Some items of note:

- The `.dat` file extension used here is not required. We use it merely to make the log and data files easily recognisable.
- Every `CREATE LOGFILE GROUP` and `ALTER LOGFILE GROUP` statement must include an `ENGINE` clause. In MySQL 5.1, the permitted values for this clause are `NDB` and `NDBCLUSTER`.

Important

In MySQL 5.1.8 and later, there can exist only one log file group at any given time.

- When you add an undo log file to a log file group using `ADD UNDOFILE 'filename'`, a file with the name *filename* is created in the `ndb_nodeid_fs` directory within the `DataDirectory` of each data node in the cluster, where *nodeid* is the node ID of the data node.
- `UNDO_BUFFER_SIZE` is limited by the amount of system memory available.
- For more information about the `CREATE LOGFILE GROUP` statement, see [Section 12.1.8, “CREATE LOGFILE GROUP Syntax”](#). For more information about `ALTER LOGFILE GROUP`, see [Section 12.1.2, “ALTER LOGFILE GROUP Syntax”](#).

2.

Now we can create a tablespace, which contains files to be used by MySQL Cluster Disk Data tables for storing their data. A tablespace is also associated with a particular log file group. When creating a new tablespace, you must specify the log file group which it is to use for undo logging; you must also specify a data file. You can add more data files to the tablespace after the tablespace is created; it is also possible to drop data files from a tablespace (an example of dropping data files is provided later in this section).

Assume that we wish to create a tablespace named `ts_1` which uses `lg_1` as its log file group. This tablespace is to contain two data files named `data_1.dat` and `data_2.dat`, whose initial sizes are 32 MB and 48 MB, respectively. (The default value for `INITIAL_SIZE` is 128 MB.) We can do this using two SQL statements, as shown here:

```
CREATE TABLESPACE ts_1
  ADD DATAFILE 'data_1.dat'
  USE LOGFILE GROUP lg_1
  INITIAL_SIZE 32M
  ENGINE NDB;

ALTER TABLESPACE ts_1
  ADD DATAFILE 'data_2.dat'
  INITIAL_SIZE 48M
  ENGINE NDB;
```

The `CREATE TABLESPACE` statement creates a tablespace `ts_1` with the data file `data_1.dat`, and associates `ts_1` with log file group `lg_1`. The `ALTER TABLESPACE` adds the second data file (`data_2.dat`).

Some items of note:

- As is the case with the filenames used here for undo log files, there is no special significance for the `.dat` file extension; it is used merely for easy recognition.
- All `CREATE TABLESPACE` and `ALTER TABLESPACE` statements must contain an `ENGINE` clause; only tables using the same storage engine as the tablespace can be created in the tablespace. In MySQL 5.1, the only permitted values for this clause are `NDB` and `NDBCLUSTER`.

For more information about the `CREATE TABLESPACE` and `ALTER TABLESPACE` statements, see [Section 12.1.11, “CREATE TABLESPACE Syntax”](#), and [Section 12.1.5, “ALTER TABLESPACE Syntax”](#).

3.

Now it is possible to create a table whose non-indexed columns are stored on disk in the tablespace `ts_1`:

```
CREATE TABLE dt_1 (
  member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  last_name VARCHAR(50) NOT NULL,
  first_name VARCHAR(50) NOT NULL,
  dob DATE NOT NULL,
  joined DATE NOT NULL,
  INDEX(last_name, first_name)
)
TABLESPACE ts_1 STORAGE DISK
ENGINE NDB;
```

The `TABLESPACE ... STORAGE DISK` clause tells the `NDB Cluster` storage engine to use tablespace `ts_1` for disk data storage.

Note

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

Beginning with MySQL 5.1.19-ndb-6.2.5 and MySQL 5.1.20-ndb-6.3.2, it is also possible to specify whether an individual column is stored on disk or in memory by using a **STORAGE** clause as part of the column's definition in a **CREATE TABLE** or **ALTER TABLE** statement. **STORAGE DISK** causes the column to be stored on disk, and **STORAGE MEMORY** causes in-memory storage to be used. See Section 12.1.10, “**CREATE TABLE Syntax**”, for more information.

The following information applies to all MySQL Cluster users.

Once table `ts_1` has been created as shown, you can perform **INSERT**, **SELECT**, **UPDATE**, and **DELETE** statements on it just as you would with any other MySQL table.

For table `dt_1` as it has been defined here, only the `dob` and `joined` columns are stored on disk. This is because there are indexes on the `id`, `last_name`, and `first_name` columns, and so data belonging to these columns is stored in RAM. In MySQL 5.1, only non-indexed columns can be held on disk; indexes and indexed column data continue to be stored in memory. This trade-off between the use of indexes and conservation of RAM is something you must keep in mind as you design Disk Data tables.

Performance note. The performance of a cluster using Disk Data storage is greatly improved if Disk Data files are kept on a separate physical disk from the data node filesystem. This must be done for each data node in the cluster to derive any noticeable benefit.

You may use absolute and relative filesystem paths with **ADD UNDOFILE** and **ADD DATAFILE**. Relative paths are calculated relative to the data node's data directory.

A log file group, a tablespace, and any Disk Data tables using these must be created in a particular order. The same is true for dropping any of these objects:

- A log file group cannot be dropped, so long as any tablespaces are using it.
- A tablespace cannot be dropped as long as it contains any data files.
- You cannot drop any data files from a tablespace as long as there remain any tables which are using the tablespace.
- Beginning with MySQL 5.1.12, it is no longer possible to drop files created in association with a different tablespace than the one with which the files were created. ([Bug#20053](#))

For example, to drop all the objects created so far in this section, you would use the following statements:

```
mysql> DROP TABLE dt_1;

mysql> ALTER TABLESPACE ts_1
-> DROP DATAFILE 'data_2.dat'
-> ENGINE NDB;

mysql> ALTER TABLESPACE ts_1
-> DROP DATAFILE 'data_1.dat'
-> ENGINE NDB;

mysql> DROP TABLESPACE ts_1
-> ENGINE NDB;

mysql> DROP LOGFILE GROUP lg_1
-> ENGINE NDB;
```

These statements must be performed in the order shown, except that the two **ALTER TABLESPACE ... DROP DATAFILE** statements may be executed in either order.

You can obtain information about data files used by Disk Data tables by querying the **FILES** table in the **INFORMATION_SCHEMA** database. An extra “**NULL** row” was added to this table in MySQL 5.1.14 for providing additional information about undo log files. For more information and examples of use, see Section 23.21, “**The INFORMATION_SCHEMA FILES Table**”.

16.11.2. Disk Data Storage Requirements

The following items apply to Disk Data storage requirements:

- Variable-length columns of Disk Data tables take up a fixed amount of space. For each row, this is equal to the space required to store the largest possible value for that column.

For general information about calculating these values, see [Section 10.5, “Data Type Storage Requirements”](#).

- In a Disk Data table, the first 256 bytes of a [TEXT](#) or [BLOB](#) column are stored in memory; only the remainder is stored on disk.

Important

Starting the cluster with the `--initial` option does *not* remove Disk Data files. You must remove these manually prior to performing an initial restart of the cluster.

16.11.3. Disk Data Configuration Parameters

Configuration parameters affecting Disk Data behaviour include the following:

- [DiskPageBufferMemory](#)

This determines the amount of space used for caching pages on disk, and is set in the `[NDBD]` or `[NDBD DEFAULT]` section of the `config.ini` file. It is measured in bytes. Each page takes up 32 KB. This means that Cluster Disk Data storage always uses $N * 32$ KB memory where N is some non-negative integer.

- [SharedGlobalMemory](#)

This determines the amount of memory that is used for log buffers, disk operations (such as page requests and wait queues), and metadata for tablespaces, log file groups, [UNDO](#) files, and data files. It can be set in the `[NDBD]` or `[NDBD DEFAULT]` section of the `config.ini` configuration file, and is measured in bytes.

The default value is `20M`.

Note

The `OPTIMIZE TABLE` statement does not have any effect on Disk Data tables.

16.12. Using High-Speed Interconnects with MySQL Cluster

Even before design of [NDB Cluster](#) began in 1996, it was evident that one of the major problems to be encountered in building parallel databases would be communication between the nodes in the network. For this reason, [NDB Cluster](#) was designed from the very beginning to allow for the use of a number of different data transport mechanisms. In this Manual, we use the term *transporter* for these.

The MySQL Cluster codebase includes support for four different transporters:

- *TCP/IP using 100 Mbps or gigabit Ethernet*, as discussed in [Section 16.4.4.7, “Cluster TCP/IP Connections”](#).
- *Direct (machine-to-machine) TCP/IP*; although this transporter uses the same TCP/IP protocol as mentioned in the previous item, it requires setting up the hardware differently and is configured differently as well. For this reason, it is considered a separate transport mechanism for MySQL Cluster. See [Section 16.4.4.8, “TCP/IP Connections Using Direct Connections”](#), for details.
- *Shared memory (SHM)*. For more information about SHM, see [Section 16.4.4.9, “Shared-Memory Connections”](#).
- *Scalable Coherent Interface (SCI)*, as described in the next section of this chapter, [Section 16.4.4.10, “SCI Transport Connections”](#).

Most users today employ TCP/IP over Ethernet because it is ubiquitous. TCP/IP is also by far the best-tested transporter for use with MySQL Cluster.

We are working to make sure that communication with the `ndbd` process is made in “chunks” that are as large as possible because this benefits all types of data transmission.

For users who desire it, it is also possible to use cluster interconnects to enhance performance even further. There are two ways to achieve this: Either a custom transporter can be designed to handle this case, or you can use socket implementations that bypass the TCP/IP stack to one extent or another. We have experimented with both of these techniques using the SCI (Scalable Coherent Interface) technology developed by [Dolphin](#).

16.12.1. Configuring MySQL Cluster to use SCI Sockets

In this section, we show how to adapt a cluster configured for normal TCP/IP communication to use SCI Sockets instead. This documentation is based on SCI Sockets version 2.3.0 as of 01 October 2004.

Prerequisites. Any machines with which you wish to use SCI Sockets must be equipped with SCI cards.

No special builds (other than the `-max` builds) are needed for SCI Sockets because it uses normal TCP/IP socket calls which are already available in MySQL Cluster. However, SCI Sockets are currently supported only on the Linux 2.4 and 2.6 kernels. For other operating systems, you can use SCI Transporters, but this requires that the server be built using `--with-ndb-sci=/opt/DIS`.

Prior to MySQL 5.1.20, there were issues with building MySQL Cluster with SCI support (see [Bug#25470](#)), but these have been resolved due to work contributed by Dolphin International. SCI Sockets are now correctly supported for MySQL Cluster using the `-max` builds, and versions of MySQL Cluster with SCI Transporter support can be built using either of `compile-amd64-max-sci` or `compile-pentium64-max-sci`. Both of these build scripts can be found in the `BUILD` directory of the MySQL 5.1 source; it should not be difficult to adapt them for other platforms.

There are essentially four requirements for SCI Sockets:

- Building the SCI Socket libraries.
- Installation of the SCI Socket kernel libraries.
- Installation of one or two configuration files.
- The SCI Socket kernel library must be enabled either for the entire machine or for the shell where the MySQL Cluster processes are started.

This process needs to be repeated for each machine in the cluster where you plan to use SCI Sockets for inter-node communication.

Two packages need to be retrieved to get SCI Sockets working:

- The source code package containing the DIS support libraries for the SCI Sockets libraries.
- The source code package for the SCI Socket libraries themselves.

Currently, these are available only in source code format. The latest versions of these packages at the time of this writing were available as (respectively) `DIS_GPL_2_5_0_SEP_10_2004.tar.gz` and `SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz`. You should be able to find these (or possibly newer versions) at <http://www.dolphinics.no/support/downloads.html>.

Package Installation. Once you have obtained the library packages, the next step is to unpack them into appropriate directories, with the SCI Sockets library unpacked into a directory below the DIS code. Next, you need to build the libraries. This example shows the commands used on Linux/x86 to perform this task:

```
shell> tar xzf DIS_GPL_2_5_0_SEP_10_2004.tar.gz
shell> cd DIS_GPL_2_5_0_SEP_10_2004/src/
shell> tar xzf ../../SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz
shell> cd ../adm/bin/Linux_pkgs
shell> ./make_PSB_66_release
```

It is possible to build these libraries for some 64-bit processors. To build the libraries for Opteron CPUs using the 64-bit extensions, run `make_PSB_66_X86_64_release` rather than `make_PSB_66_release`. If the build is made on an Itanium machine, you should use `make_PSB_66_IA64_release`. The X86-64 variant should work for Intel EM64T architectures but this has not yet (to our knowledge) been tested.

Once the build process is complete, the compiled libraries will be found in a zipped tar file with a name along the lines of `DIS-<operating-system>-time-date`. It is now time to install the package in the proper place. In this example we will place the installation in `/opt/DIS`. You most likely need to run the following as the system `root` user.

```
shell> cp DIS_Linux_2.4.20-8_181004.tar.gz /opt/
shell> cd /opt
shell> tar xzf DIS_Linux_2.4.20-8_181004.tar.gz
shell> mv DIS_Linux_2.4.20-8_181004 DIS
```

Network configuration. Now that all the libraries and binaries are in their proper place, we need to ensure that the SCI cards have proper node IDs within the SCI address space.

It is also necessary to decide on the network structure before proceeding. There are three types of network structures which can be used in this context:

- A simple one-dimensional ring
- One or more SCI switches with one ring per switch port
- A two- or three-dimensional torus.

Each of these topologies has its own method for providing node IDs. We discuss each of them in brief.

A simple ring uses node IDs which are non-zero multiples of 4: 4, 8, 12,...

The next possibility uses SCI switches. An SCI switch has 8 ports, each of which can support a ring. It is necessary to make sure that different rings use different node ID spaces. In a typical configuration, the first port uses node IDs below 64 (4 – 60), the next 64 node IDs (68 – 124) are assigned to the next port, and so on, with node IDs 452 – 508 being assigned to the eighth port.

Two- and three-dimensional torus network structures take into account where each node is located in each dimension, incrementing by 4 for each node in the first dimension, by 64 in the second dimension, and (where applicable) by 1024 in the third dimension. See [Dolphin's Web site](#) for more thorough documentation.

In our testing we have used switches, although most large cluster installations use 2- or 3-dimensional torus structures. The advantage provided by switches is that, with dual SCI cards and dual switches, it is possible to build with relative ease a redundant network where the average failover time on the SCI network is on the order of 100 microseconds. This is supported by the SCI transporter in MySQL Cluster and is also under development for the SCI Socket implementation.

Failover for the 2D/3D torus is also possible but requires sending out new routing indexes to all nodes. However, this requires only 100 milliseconds or so to complete and should be acceptable for most high-availability cases.

By placing cluster data nodes properly within the switched architecture, it is possible to use 2 switches to build a structure whereby 16 computers can be interconnected and no single failure can hinder more than one of them. With 32 computers and 2 switches it is possible to configure the cluster in such a manner that no single failure can cause the loss of more than two nodes; in this case, it is also possible to know which pair of nodes is affected. Thus, by placing the two nodes in separate node groups, it is possible to build a “safe” MySQL Cluster installation.

To set the node ID for an SCI card use the following command in the `/opt/DIS/sbin` directory. In this example, `-c 1` refers to the number of the SCI card (this is always 1 if there is only 1 card in the machine); `-a 0` refers to adapter 0; and `68` is the node ID:

```
shell> ./sciconfig -c 1 -a 0 -n 68
```

If you have multiple SCI cards in the same machine, you can determine which card has which slot by issuing the following command (again we assume that the current working directory is `/opt/DIS/sbin`):

```
shell> ./sciconfig -c 1 -gsn
```

This will give you the SCI card's serial number. Then repeat this procedure with `-c 2`, and so on, for each card in the machine. Once you have matched each card with a slot, you can set node IDs for all cards.

After the necessary libraries and binaries are installed, and the SCI node IDs are set, the next step is to set up the mapping from host-names (or IP addresses) to SCI node IDs. This is done in the SCI sockets configuration file, which should be saved as `/etc/sci/scisock.conf`. In this file, each SCI node ID is mapped through the proper SCI card to the hostname or IP address that it

is to communicate with. Here is a very simple example of such a configuration file:

```
#host      #nodeId
alpha      8
beta       12
192.168.10.20 16
```

It is also possible to limit the configuration so that it applies only to a subset of the available ports for these hosts. An additional configuration file `/etc/sci/scisock_opt.conf` can be used to accomplish this, as shown here:

```
# -key          -type          -values
EnablePortsByDefault  yes
EnablePort            tcp          2200
DisablePort           tcp          2201
EnablePortRange       tcp          2202 2219
DisablePortRange      tcp          2220 2231
```

Driver installation. With the configuration files in place, the drivers can be installed.

First, the low-level drivers and then the SCI socket driver need to be installed:

```
shell> cd DIS/sbin/
shell> ./drv-install add PSB66
shell> ./scisocket-install add
```

If desired, the installation can be checked by invoking a script which verifies that all nodes in the SCI socket configuration files are accessible:

```
shell> cd /opt/DIS/sbin/
shell> ./status.sh
```

If you discover an error and need to change the SCI socket configuration, it is necessary to use `ksocketconfig` to accomplish this task:

```
shell> cd /opt/DIS/util
shell> ./ksocketconfig -f
```

Testing the setup. To ensure that SCI sockets are actually being used, you can employ the `latency_bench` test program. Using this utility's server component, clients can connect to the server to test the latency of the connection. Determining whether SCI is enabled should be fairly simple from observing the latency.

Note

Before using `latency_bench`, it is necessary to set the `LD_PRELOAD` environment variable as shown later in this section.

To set up a server, use the following:

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -server
```

To run a client, use `latency_bench` again, except this time with the `-client` option:

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -client server_hostname
```

SCI socket configuration should now be complete and MySQL Cluster ready to use both SCI Sockets and the SCI transporter (see [Section 16.4.4.10, “SCI Transport Connections”](#)).

Starting the cluster. The next step in the process is to start MySQL Cluster. To enable usage of SCI Sockets it is necessary to set the environment variable `LD_PRELOAD` before starting `ndbd`, `mysqld`, and `ndb_mgmd`. This variable should point to the kernel library for SCI Sockets.

To start `ndbd` in a bash shell, do the following:

```
bash-shell> export LD_PRELOAD=/opt/DIS/lib/libkscisock.so
bash-shell> ndbd
```

In a tcsh environment the same thing can be accomplished with:

```
tcsh-shell> setenv LD_PRELOAD=/opt/DIS/lib/libkscisock.so
tcsh-shell> ndbd
```

Note

MySQL Cluster can use only the kernel variant of SCI Sockets.

16.12.2. Understanding the Impact of Cluster Interconnects

The `ndbd` process has a number of simple constructs which are used to access the data in a MySQL Cluster. We have created a very simple benchmark to check the performance of each of these and the effects which various interconnects have on their performance.

There are four access methods:

- **Primary key access.** This is access of a record through its primary key. In the simplest case, only one record is accessed at a time, which means that the full cost of setting up a number of TCP/IP messages and a number of costs for context switching are borne by this single request. In the case where multiple primary key accesses are sent in one batch, those accesses share the cost of setting up the necessary TCP/IP messages and context switches. If the TCP/IP messages are for different destinations, additional TCP/IP messages need to be set up.
- **Unique key access.** Unique key accesses are similar to primary key accesses, except that a unique key access is executed as a read on an index table followed by a primary key access on the table. However, only one request is sent from the MySQL Server, and the read of the index table is handled by `ndbd`. Such requests also benefit from batching.
- **Full table scan.** When no indexes exist for a lookup on a table, a full table scan is performed. This is sent as a single request to the `ndbd` process, which then divides the table scan into a set of parallel scans on all cluster `ndbd` processes. In future versions of MySQL Cluster, an SQL node will be able to filter some of these scans.

- **Range scan using ordered index**

When an ordered index is used, it performs a scan in the same manner as the full table scan, except that it scans only those records which are in the range used by the query transmitted by the MySQL server (SQL node). All partitions are scanned in parallel when all bound index attributes include all attributes in the partitioning key.

To check the base performance of these access methods, we have developed a set of benchmarks. One such benchmark, `testReadPerf`, tests simple and batched primary and unique key accesses. This benchmark also measures the setup cost of range scans by issuing scans returning a single record. There is also a variant of this benchmark which uses a range scan to fetch a batch of records.

In this way, we can determine the cost of both a single key access and a single record scan access, as well as measure the impact of the communication media used, on base access methods.

In our tests, we ran the base benchmarks for both a normal transporter using TCP/IP sockets and a similar setup using SCI sockets. The figures reported in the following table are for small accesses of 20 records per access. The difference between serial and batched access decreases by a factor of 3 to 4 when using 2KB records instead. SCI Sockets were not tested with 2KB records. Tests were performed on a cluster with 2 data nodes running on 2 dual-CPU machines equipped with AMD MP1900+ processors.

Access Type	TCP/IP Sockets	SCI Socket
Serial pk access	400 microseconds	160 microseconds
Batched pk access	28 microseconds	22 microseconds
Serial uk access	500 microseconds	250 microseconds
Batched uk access	70 microseconds	36 microseconds
Indexed eq-bound	1250 microseconds	750 microseconds
Index range	24 microseconds	12 microseconds

We also performed another set of tests to check the performance of SCI Sockets *vis-à-vis* that of the SCI transporter, and both of these as compared with the TCP/IP transporter. All these tests used primary key accesses either serially and multi-threaded, or multi-threaded and batched.

The tests showed that SCI sockets were about 100% faster than TCP/IP. The SCI transporter was faster in most cases compared to SCI sockets. One notable case occurred with many threads in the test program, which showed that the SCI transporter did not perform very well when used for the `mysqld` process.

Our overall conclusion was that, for most benchmarks, using SCI sockets improves performance by approximately 100% over TCP/IP, except in rare instances when communication performance is not an issue. This can occur when scan filters make up most of processing time or when very large batches of primary key accesses are achieved. In that case, the CPU processing in the `ndbd` processes becomes a fairly large part of the overhead.

Using the SCI transporter instead of SCI Sockets is only of interest in communicating between `ndbd` processes. Using the SCI transporter is also only of interest if a CPU can be dedicated to the `ndbd` process because the SCI transporter ensures that this process will never go to sleep. It is also important to ensure that the `ndbd` process priority is set in such a way that the process does not lose priority due to running for an extended period of time, as can be done by locking processes to CPUs in Linux 2.6. If such a configuration is possible, the `ndbd` process will benefit by 10–70% as compared with using SCI sockets. (The larger figures will be seen when performing updates and probably on parallel scan operations as well.)

There are several other optimized socket implementations for computer clusters, including Myrinet, Gigabit Ethernet, Infiniband and the VIA interface. We have tested MySQL Cluster so far only with SCI sockets. See [Section 16.12.1, “Configuring MySQL Cluster to use SCI Sockets”](#) for information on how to set up SCI sockets using ordinary TCP/IP for MySQL Cluster.

16.13. Known Limitations of MySQL Cluster

In the sections that follow, we discuss known limitations in MySQL 5.1 Cluster and MySQL Cluster 5.1 Carrier Grade Edition releases as compared with the features available when using the `MyISAM` and `InnoDB` storage engines. Currently, there are no plans to address these in coming releases of MySQL 5.1 or MySQL Cluster 5.1 Carrier Grade Edition; however, we will attempt to supply fixes for these issues in subsequent release series. If you check the “Cluster” category in the MySQL bugs database at <http://bugs.mysql.com>, you can find known bugs which (if marked “5.1”) we intend to correct in upcoming releases of MySQL 5.1. Some of these issues may be addressed in MySQL Cluster 5.1 Carrier Grade Edition and the fixes included in a future mainline MySQL Cluster version.

This information is intended to be complete with respect to the conditions just set forth. You can report any discrepancies that you encounter to the MySQL bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#). If we do not plan to fix the problem in MySQL 5.1, we will add it to the list.

See [Section 16.13.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1”](#) for a list of issues in MySQL Cluster in MySQL 5.0 that have been resolved in the current version.

Note

Limitations and other issues specific to MySQL Cluster Replication are described in [Section 16.10.3, “Known Issues in MySQL Cluster Replication”](#).

16.13.1. Non-Compliance In SQL Syntax

Some SQL statements relating to certain MySQL features produce errors when used with `NDB` tables, as described in the following list:

- **Temporary tables.** Temporary tables are not supported. Trying either to create a temporary table that uses the `NDB` storage engine or to alter an existing temporary table to use `NDB` fails with the error `TABLE STORAGE ENGINE 'NDBCLUSTER' DOES NOT SUPPORT THE CREATE OPTION 'TEMPORARY'`.
- **Indexes and keys in `NDB` tables.** Keys and indexes on MySQL Cluster tables are subject to the following limitations:
 - **TEXT and BLOB columns.** You cannot create indexes on `NDB` table columns that use any of the `TEXT` or `BLOB` data types.
 - **FULLTEXT indexes.** The `NDB` storage engine does not support `FULLTEXT` indexes, which are possible for `MyISAM` tables only.

However, you can create indexes on `VARCHAR` columns of `NDB` tables.

- **BIT columns.** A `BIT` column cannot be a primary key, unique key, or index, nor can it be part of a composite primary key, unique key, or index.
- **AUTO_INCREMENT columns.** Like other MySQL storage engines, the `NDB` storage engine can handle a maximum of one `AUTO_INCREMENT` column per table. However, in the case of a Cluster table with no explicit primary key, an

`AUTO_INCREMENT` column is automatically defined and used as a “hidden” primary key. For this reason, you cannot define a table that has an explicit `AUTO_INCREMENT` column unless that column is also declared using the `PRIMARY KEY` option. Attempting to create a table with an `AUTO_INCREMENT` column that is not the table's primary key, and using the `NDB` storage engine, fails with an error.

- **MySQL Cluster and geometry data types.** Geometry datatypes (`WKT` and `WKB`) are supported in `NDB` tables in MySQL 5.1. However, spatial indexes are not supported.
- **Creating `NDB` tables with user-defined partitioning.** Support for user-defined partitioning for MySQL Cluster in MySQL 5.1 is restricted to `[LINEAR] KEY` partitioning. Beginning with MySQL 5.1.12, using any other partitioning type with `ENGINE=NDB` or `ENGINE=NDBCLUSTER` in a `CREATE TABLE` statement results in an error.
- **Default partitioning scheme.** As of MySQL 5.1.6, all Cluster tables are by default partitioned by `KEY` using the table's primary key as the partitioning key. If no primary key is explicitly set for the table, the “hidden” primary key automatically created by the `NDB` storage engine is used instead. For additional discussion of these and related issues, see [Section 17.2.4, “KEY Partitioning”](#).
- **`DROP PARTITION` not supported.** It is not possible to drop partitions from `NDB` tables using `ALTER TABLE ... DROP PARTITION`. The other partitioning extensions to `ALTER TABLE` — `ADD PARTITION`, `REORGANIZE PARTITION`, and `COALESCE PARTITION` — are supported for Cluster tables, but use copying and so are not optimised. See [Section 17.3.1, “Management of RANGE and LIST Partitions”](#) and [Section 12.1.4, “ALTER TABLE Syntax”](#).
- **Row-based replication.** When using row-based replication with MySQL Cluster, binary logging cannot be disabled. That is, the `NDB` storage engine ignores the value of `SQL_LOG_BIN`. (Bug#16680)

16.13.2. Limits and Differences from Standard MySQL Limits

In this section, we list limits found in MySQL Cluster that either differ from limits found in, or that are not found in, standard MySQL.

- **Memory usage and recovery.** Memory consumed when data is inserted into an `NDB` table is not automatically recovered when deleted, as it is with other storage engines. Instead, the following rules hold true:
 - A `DELETE` statement on an `NDB` table makes the memory formerly used by the deleted rows available for re-use by inserts on the same table only. This memory cannot be used by other `NDB` tables.
 - A `DROP TABLE` or `TRUNCATE` operation on an `NDB` table frees the memory that was used by this table for re-use by any `NDB` table, either by the same table or by another `NDB` table.

Note

Recall that `TRUNCATE` drops and re-creates the table. See [Section 12.2.9, “TRUNCATE Syntax”](#).

Memory freed by `DELETE` operations but still allocated to a specific table can also be made available for general re-use by performing a rolling restart of the cluster. See [Section 16.5.1, “Performing a Rolling Restart of the Cluster”](#).

- **Limits imposed by the cluster's configuration.** A number of hard limits exist which are configurable, but available main memory in the cluster sets limits. See the complete list of configuration parameters in [Section 16.4.4, “Configuration File”](#). Most configuration parameters can be upgraded online. These hard limits include:
 - Database memory size and index memory size (`DataMemory` and `IndexMemory`, respectively).

`DataMemory` is allocated as 32KB pages. As each `DataMemory` page is used, it is assigned to a specific table; once allocated, this memory cannot be freed except by dropping the table.

See [Section 16.4.4.5, “Defining Data Nodes”](#), for further information about `DataMemory` and `IndexMemory`.
 - The maximum number of operations that can be performed per transaction is set using the configuration parameters `MaxNoOfConcurrentOperations` and `MaxNoOfLocalOperations`.

Note

Bulk loading, `TRUNCATE TABLE`, and `ALTER TABLE` are handled as special cases by running multiple transactions, and so are not subject to this limitation.

- Different limits related to tables and indexes. For example, the maximum number of ordered indexes per table is determined by `MaxNoOfOrderedIndexes`.
- **Node and data object maximums.** The following limits apply to numbers of cluster nodes and metadata objects:
 - The maximum number of data nodes is 48.
A data node cannot have a node ID greater than 49.
 - The total maximum number of nodes in a MySQL Cluster is 63. This number includes all SQL nodes (MySQL Servers), API nodes (applications accessing the cluster other than MySQL servers), data nodes, and management servers.
 - The maximum number of metadata objects in MySQL 5.1 Cluster and MySQL Cluster 5.1 Carrier Grade Edition is 20320. This limit is hard-coded.

Note

MySQL Cluster 5.1 Carrier Grade Edition users should see Section 16.13.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1”, for more information.

16.13.3. Limits Relating to Transaction Handling

A number of limitations exist in MySQL Cluster with regard to the handling of transactions. These include the following:

- **Transaction isolation level.** The `NDBCLUSTER` storage engine supports only the `READ COMMITTED` transaction isolation level. (`InnoDB`, for example, supports `READ COMMITTED`, `READ UNCOMMITTED`, `REPEATABLE READ`, and `SERIALIZABLE`.) See Section 16.8.5, “Backup Troubleshooting”, for information on how this can affect backing up and restoring Cluster databases.)

Important

If a `SELECT` from a Cluster table includes a `BLOB` or `TEXT` column, the `READ COMMITTED` transaction isolation level is converted to a read with read lock. This is done to guarantee consistency, due to the fact that parts of the values stored in columns of these types are actually read from a separate table.

- **Rollbacks.** There is no partial rollback of transactions. A duplicate key or similar error rolls back the entire transaction.
- **Transactions and memory usage.** As noted elsewhere in this chapter, MySQL Cluster does not handle large transactions well; it is better to perform a number of small transactions with a few operations each than to attempt a single large transaction containing a great many operations. Among other considerations, large transactions require very large amounts of memory. Because of this, the transactional behaviour of a number of MySQL statements is effected as described in the following list:
 - `TRUNCATE` is not transactional when used on `NDB` tables. If a `TRUNCATE` fails to empty the table, then it must be re-run until it is successful.
 - `DELETE FROM` (even with no `WHERE` clause) is transactional. For tables containing a great many rows, you may find that performance is improved by using several `DELETE FROM ... LIMIT ...` statements to “chunk” the delete operation. If your objective is to empty the table, then you may wish to use `TRUNCATE` instead.
 - **`LOAD DATA` statements.** `LOAD DATA INFILE` is not transactional when used on `NDB` tables.

Important

When executing a `LOAD DATA INFILE` statement, the `NDB` engine performs commits at irregular intervals that enable better utilization of the communication network. It is not possible to know ahead of time when such commits take place. `LOAD DATA FROM MASTER` is not supported in MySQL Cluster.

- **`ALTER TABLE` and transactions.** When copying an `NDB` table as part of an `ALTER TABLE`, the creation of the copy is non-

transactional. (In any case, this operation is rolled back when the copy is deleted.)

16.13.4. Error Handling

Starting, stopping, or restarting a node may give rise to temporary errors causing some transactions to fail. These include the following cases:

- **Temporary errors.** When first starting a node, it is possible that you may see Error 1204 `TEMPORARY FAILURE, DISTRIBUTION CHANGED` and similar temporary errors.
- **Errors due to node failure.** The stopping or failure of any data node can result in a number of different node failure errors. (However, there should be no aborted transactions when performing a planned shutdown of the cluster.)

In either of these cases, any errors that are generated must be handled within the application. This should be done by retrying the transaction.

See also [Section 16.13.2, “Limits and Differences from Standard MySQL Limits”](#).

16.13.5. Limits Associated with Database Objects

Some database objects such as tables and indexes have different limitations when using the `NDBCLUSTER` storage engine:

- **Identifiers.** Database names, table names and attribute names cannot be as long in `NDB` tables as when using other table handlers. Attribute names are truncated to 31 characters, and if not unique after truncation give rise to errors. Database names and table names can total a maximum of 122 characters. In other words, the maximum length for an `NDB` table name is 122 characters, less the number of characters in the name of the database of which that table is a part.
- **Number of tables.** The maximum number of `NDB` tables is limited to 20320.
- **Attributes per table.** The maximum number of attributes (that is, columns and indexes) per table is limited to 128.
- **Attributes per key.** The maximum number of attributes per key is 32.
- **Row size.** The maximum permitted size of any one row is 8KB. Note that each `BLOB` or `TEXT` column contributes $256 + 8 = 264$ bytes towards this total.

16.13.6. Unsupported Or Missing Features

A number of features supported by other storage engines are not supported for `NDB` tables. Trying to use any of these features in MySQL Cluster does not cause errors in or of itself; however, errors may occur in applications that expects the features to be supported or enforced:

- **Foreign key constraints.** The foreign key construct is ignored, just as it is in `MyISAM` tables.
- **OPTIMIZE operations.** `OPTIMIZE` operations are not supported.
- **LOAD TABLE ... FROM MASTER.** `LOAD TABLE ... FROM MASTER` is not supported.
- **Savepoints and rollbacks.** Savepoints and rollbacks to savepoints are ignored as in `MyISAM`.
- **Durability of commits.** There are no durable commits on disk. Commits are replicated, but there is no guarantee that logs are flushed to disk on commit.
- **Replication.** Statement-based replication is not supported. Use `--binlog-format=ROW` (or `--binlog-format=MIXED`) when setting up cluster replication. See [Section 16.10, “MySQL Cluster Replication”](#), for more information.

■ Note

See [Section 16.13.3, “Limits Relating to Transaction Handling”](#), for more information relating to limitations on transaction handling in [NDB](#).

16.13.7. Limitations Relating to Performance

The following performance issues are specific to or especially pronounced in MySQL Cluster:

- **Range scans.** There are query performance issues due to sequential access to the [NDB](#) storage engine; it is also relatively more expensive to do many range scans than it is with either [MyISAM](#) or [InnoDB](#).
- **Reliability of [Records in range](#).** The [Records in range](#) statistic is available but is not completely tested or officially supported. This may result in non-optimal query plans in some cases. If necessary, you can employ [USE INDEX](#) or [FORCE INDEX](#) to alter the execution plan. See [Section 12.2.7.2, “Index Hint Syntax”](#), for more information on how to do this.
- **Unique hash indexes.** Unique hash indexes created with [USING HASH](#) cannot be used for accessing a table if [NULL](#) is given as part of the key.

16.13.8. Issues Exclusive to MySQL Cluster

The following are limitations specific to the [NDBCLUSTER](#) storage engine:

- **Machine architecture.** All machines used in the cluster must have the same architecture. That is, all machines hosting nodes must be either big-endian or little-endian, and you cannot use a mixture of both. For example, you cannot have a management node running on a PowerPC which directs a data node that is running on an x86 machine. This restriction does not apply to machines simply running [mysql](#) or other clients that may be accessing the cluster's SQL nodes.
- **Adding and dropping of data nodes.** Online adding or dropping of data nodes is not currently possible. In such cases, the entire cluster must be restarted.
- **Binary logging.** MySQL Cluster has the following limitations or restrictions with regard to binary logging:
 - [SQL_LOG_BIN](#) has no effect on data operations; however, it is supported for schema operations.
 - MySQL Cluster cannot produce a binlog for tables having [BLOB](#) columns but no primary key.
 - Only the following schema operations are logged in a cluster binlog which is *not* on the [mysqld](#) executing the statement:
 - [CREATE TABLE](#)
 - [ALTER TABLE](#)
 - [DROP TABLE](#)
 - [CREATE DATABASE / CREATE SCHEMA](#)
 - [DROP DATABASE / DROP SCHEMA](#)
 - [CREATE TABLESPACE](#)
 - [ALTER TABLESPACE](#)
 - [DROP TABLESPACE](#)
 - [CREATE LOGFILE GROUP](#)
 - [ALTER LOGFILE GROUP](#)
 - [DROP LOGFILE GROUP](#)

See also [Section 16.13.10, “Limitations Relating to Multiple Cluster Nodes”](#).

16.13.9. Limitations Relating to Disk Data Storage

- Disk data objects are subject to the following maximums:
 - Maximum number of tablespaces: 2^{32} (4294967296)
 - Maximum number of data files per tablespace: 2^{16} (65535)
 - The theoretical maximum number of extents per tablespace data file is 2^{16} (65525); however, for practical purposes, the recommended maximum number of extents per data file is 2^8 (32768).
 - Maximum data file size: The theoretical limit is 64G; however, in MySQL 5.1, the practical upper limit is 32G. This is equivalent to 32768 extents of 1M each.

The minimum and maximum possible sizes of extents for tablespace data files are 32K and 2G, respectively. See [Section 12.1.11, “CREATE TABLESPACE Syntax”](#), for more information.
- Use of Disk Data tables is not supported when running the cluster in diskless mode. Beginning with MySQL 5.1.12, it is disallowed altogether. ([Bug#20008](#))

16.13.10. Limitations Relating to Multiple Cluster Nodes

Multiple SQL nodes. The following are issues relating to the use of multiple MySQL servers as MySQL Cluster SQL nodes, and are specific to the [NDBCLUSTER](#) storage engine:

- **ALTER TABLE operations.** [ALTER TABLE](#) is not fully locking when running multiple MySQL servers (no distributed table lock).
- **DDL operations.** DDL operations (such as [CREATE TABLE](#) or [ALTER TABLE](#)) are not safe from data node failures. If a data node fails while trying to perform one of these, the data dictionary is locked and no further DDL statements can be executed without restarting the cluster.

Multiple management nodes. When using multiple management servers:

- You must give nodes explicit IDs in connectstrings because automatic allocation of node IDs does not work across multiple management servers.
- You must take extreme care to have the same configurations for all management servers. No special checks for this are performed by the cluster.

Multiple data node processes. While it is possible to run multiple cluster processes concurrently on a single host, it is not always advisable to do so for reasons of performance and high availability, as well as other considerations. In particular, in MySQL 5.1 or MySQL Cluster 5.1 Carrier Grade Edition, we do not support for production use any MySQL Cluster deployment in which more than one [ndbd](#) process is run on a single physical machine.

Note

We may support multiple data nodes per host in a future MySQL release, following additional testing. However, in MySQL 5.1 and MySQL Cluster 5.1 Carrier Grade Edition, such configurations can be considered experimental only.

Multiple network addresses. Multiple network addresses per data node are not supported. Use of these is liable to cause problems: In the event of a data node failure, an SQL node waits for confirmation that the data node went down but never receives it because another route to that data node remains open. This can effectively make the cluster inoperable.

Note

It is possible to use multiple network hardware *interfaces* (such as Ethernet cards) for a single data node, but these must be bound to the same address. This also means that it not possible to use more than one [\[TCP\]](#) section per connection in the [config.ini](#) file. See [Section 16.4.4.7, “Cluster TCP/IP Connections”](#), for more information.

16.13.11. Previous MySQL Cluster Issues Resolved in MySQL 5.1

A number of limitations and related issues existing in earlier versions of MySQL Cluster have been resolved over the course of development of MySQL 5.1 or MySQL Cluster 5.1 Carrier Grade Edition:

- **Variable-length column support.** The **NDB Cluster** storage engine now supports variable-length column types for in-memory tables.

Previously, for example, any Cluster table having one or more **VARCHAR** fields which contained only relatively small values, much more memory and disk space were required when using the **NDBCLUSTER** storage engine than would have been the case for the same table and data using the **MyISAM** engine. In other words, in the case of a **VARCHAR** column, such a column required the same amount of storage as a **CHAR** column of the same size. In MySQL 5.1, this is no longer the case for in-memory tables, where storage requirements for variable-length column types such as **VARCHAR** and **BINARY** are comparable to those for these column types when used in **MyISAM** tables (see [Section 10.5, “Data Type Storage Requirements”](#)).

Important

For MySQL Cluster Disk Data tables, the fixed-width limitation continues to apply. See [Section 16.11, “MySQL Cluster Disk Data Tables”](#).

- **Replication with MySQL Cluster.** It is now possible to use MySQL replication with Cluster databases. For details, see [Section 16.10, “MySQL Cluster Replication”](#).
Circular Replication. Circular replication is supported for MySQL Cluster beginning with MySQL 5.1.18. See [Section 16.10.3, “Known Issues in MySQL Cluster Replication”](#).
- **auto_increment_increment and auto_increment_offset.** The **auto_increment_increment** and **auto_increment_offset** server system variables are supported for Cluster replication beginning with MySQL 5.1.20.
- **Database autodiscovery and online schema changes.** Autodiscovery of databases is now supported for multiple MySQL servers accessing the same MySQL Cluster, provided that a given **mysqld** is already running and is connected to the cluster at the time that the database is created on a different **mysqld**.

What this means is that if a **mysqld** process first connects to the cluster after a database named **db_name** has been created, you should issue a **CREATE SCHEMA db_name** statement on the “new” MySQL server when it first accesses that MySQL Cluster. Once this has been done, the “new” **mysqld** should be able to detect any tables in that database tables without errors.

This also means that online schema changes in **NDB** tables are now possible. That is, the result of operations such as **ALTER TABLE** and **CREATE INDEX** performed on one SQL node in the cluster are now visible to the cluster's other SQL nodes without any additional action being taken.

- **Backup and restore between architectures.** Beginning with MySQL 5.1.10, it is possible to perform a Cluster backup and restore between different architectures. Previously — for example — you could not back up a cluster running on a big-endian platform and then restore from that backup to a cluster running on a little-endian system. ([Bug#19255](#))
- **Character set directory.** Beginning with MySQL 5.1.10, it is possible to install MySQL with Cluster support to a non-default location and change the search path for font description files using either the **--basedir** or **--character-sets-dir** options. (Previously, **ndbd** in MySQL 5.1 searched only the default path — typically **/usr/local/mysql/share/mysql/charsets** — for character sets.)
- In MySQL 5.1, it is no longer necessary, when running multiple management servers, to restart all the cluster's data nodes to enable the management nodes to see one another.
- **Length of CREATE TABLE statements.** **CREATE TABLE** statements may be no more than 4096 characters in length. *This limitation affects MySQL 5.1.6, 5.1.7, and 5.1.8 only.* (See [Bug#17813](#))
- **IGNORE and REPLACE functionality.** In MySQL 5.1.7 and earlier, **INSERT IGNORE**, **UPDATE IGNORE**, and **REPLACE** were supported only for primary keys, but not for unique keys. It was possible to work around this issue by removing the constraint, then dropping the unique index, performing any inserts, and then adding the unique index again.

This limitation was removed for **INSERT IGNORE** and **REPLACE** in MySQL 5.1.8. (See [Bug#17431](#).)

- **AUTO_INCREMENT columns.** In MySQL 5.1.10 and earlier versions, the maximum number of tables having **AUTO_INCREMENT** columns — including those belonging to hidden primary keys — was 2048.

This limitation was lifted in MySQL 5.1.11.

MySQL Cluster 5.1 Carrier Grade Edition

The following information applies to users of MySQL Cluster 5.1 Carrier Grade Edition only. For more information about MySQL Cluster 5.1 Carrier Grade Edition, see Section 16.2, “MySQL Cluster 5.1 Carrier Grade Edition”.

Maximum number of cluster nodes. Prior to MySQL 5.1.15-ndb-6.1.1, the total maximum number of nodes in a MySQL Cluster was 63, including all SQL nodes (MySQL Servers), API nodes (applications accessing the cluster other than MySQL servers), data nodes, and management servers.

Starting with MySQL 5.1.15-ndb-6.1.1, up to 255 API nodes (including MySQL servers acting as cluster SQL nodes) are supported by a single cluster. The total number of data nodes and management nodes beginning with this version is 63, of which up to 48 can be data nodes.

The limitation that a data node cannot have a node ID greater than 49 continues to apply in MySQL Cluster 5.1 Carrier Grade Edition.

The following information applies to all MySQL Cluster users.

16.14. MySQL Cluster Development Roadmap

In this section, we discuss changes in the implementation of MySQL Cluster in MySQL 5.1 and MySQL Cluster 5.1 Carrier Grade Edition as compared to MySQL 5.0.

There are a number of significant changes in the implementation of the **NDB Cluster** storage engine in MySQL 5.1 as compared to that in MySQL 5.0. For an overview of these changes, see [Section 16.14.1, “MySQL Cluster Changes in MySQL 5.1”](#)

16.14.1. MySQL Cluster Changes in MySQL 5.1

A number of new features for MySQL Cluster have been implemented in MySQL 5.1:

- **Integration of MySQL Cluster into MySQL Replication.** This makes it possible to update from any MySQL Server in the cluster and still have the MySQL Replication handled by one of the MySQL Servers in the cluster, with the state of the slave side remaining consistent with the cluster acting as the master.

See [Section 16.10, “MySQL Cluster Replication”](#).

- **Support for disk-based records.** Records on disk are now supported. Indexed columns, including the primary key hash index, must still be stored in RAM; however, all other columns can be stored on disk.

See [Section 16.11, “MySQL Cluster Disk Data Tables”](#).

- **Variable-sized records.** A column defined as **VARCHAR(255)** currently uses 260 bytes of storage independent of what is stored in any particular record. In MySQL 5.1 Cluster tables, only the portion of the column actually taken up by the record will be stored. This will make possible a reduction in space requirements for such columns by a factor of 5 in many cases.
- **User-defined partitioning.** Users can define partitions based on columns that are part of the primary key. It is possible to partition **NDB** tables based on **KEY** and **LINEAR KEY** schemes. This feature is also available for many other MySQL storage engines, which support additional partitioning types that are not available with **NDB Cluster** tables.

For additional general information about user-defined partitioning in MySQL 5.1, see [Chapter 17, Partitioning](#). Specifics of partitioning types are discussed in [Section 17.2, “Partition Types”](#).

The MySQL Server can also determine whether it is possible to “prune away” some of the partitions from the **WHERE** clause. See [Section 17.4, “Partition Pruning”](#).

- **Autodiscovery of table schema changes.** In MySQL 5.1, you no longer need to issue **FLUSH TABLES** or a “dummy” **SELECT** in order for new **NDB** tables or changes made to schemas of existing **NDB** tables on one SQL node to be visible on the cluster's other SQL nodes.

Note

When creating a new database, it is still necessary to issue a `CREATE DATABASE` or `CREATE SCHEMA` statement on each SQL node in the cluster.

See [Section 16.13.11, “Previous MySQL Cluster Issues Resolved in MySQL 5.1”](#), for more information.

16.15. MySQL Cluster Glossary

The following terms are useful to an understanding of MySQL Cluster or have specialized meanings when used in relation to it.

- **Cluster.** In its generic sense, a cluster is a set of computers functioning as a unit and working together to accomplish a single task.
NDB Cluster. This is the storage engine used in MySQL to implement data storage, retrieval, and management distributed among several computers.
MySQL Cluster. This refers to a group of computers working together using the **NDB** storage engine to support a distributed MySQL database in a *shared-nothing architecture* using *in-memory storage*.
- **Configuration files.** Text files containing directives and information regarding the cluster, its hosts, and its nodes. These are read by the cluster's management nodes when the cluster is started. See [Section 16.4.4, “Configuration File”](#), for details.
- **Backup.** A complete copy of all cluster data, transactions and logs, saved to disk or other long-term storage.
- **Restore.** Returning the cluster to a previous state, as stored in a backup.
- **Checkpoint.** Generally speaking, when data is saved to disk, it is said that a checkpoint has been reached. More specific to Cluster, it is a point in time where all committed transactions are stored on disk. With regard to the **NDB** storage engine, there are two types of checkpoints which work together to ensure that a consistent view of the cluster's data is maintained:
 - **Local Checkpoint (LCP).** This is a checkpoint that is specific to a single node; however, LCP's take place for all nodes in the cluster more or less concurrently. An LCP involves saving all of a node's data to disk, and so usually occurs every few minutes. The precise interval varies, and depends upon the amount of data stored by the node, the level of cluster activity, and other factors.
 - **Global Checkpoint (GCP).** A GCP occurs every few seconds, when transactions for all nodes are synchronized and the redo-log is flushed to disk.
- **Cluster host.** A computer making up part of a MySQL Cluster. A cluster has both a *physical* structure and a *logical* structure. Physically, the cluster consists of a number of computers, known as *cluster hosts* (or more simply as *hosts*. See also **Node** and **Node group** below.
- **Node.** This refers to a logical or functional unit of MySQL Cluster, and is sometimes also referred to as a *cluster node*. In the context of MySQL Cluster, we use the term “node” to indicate a *process* rather than a physical component of the cluster. There are three node types required to implement a working MySQL Cluster:
 - **Management nodes.** Manages the other nodes within the MySQL Cluster. It provides configuration data to the other nodes; starts and stops nodes; handles network partitioning; creates backups and restores from them, and so forth.
 - **SQL nodes.** Instances of MySQL Server which serve as front ends to data kept in the cluster's **data nodes**. Clients desiring to store, retrieve, or update data can access an SQL node just as they would any other MySQL Server, employing the usual MySQL authentication methods and APIs; the underlying distribution of data between node groups is transparent to users and applications. SQL nodes access the cluster's databases as a whole without regard to the data's distribution across different data nodes or cluster hosts.
 - **Data nodes.** These nodes store the actual data. Table data fragments are stored in a set of node groups; each node group stores a different subset of the table data. Each of the nodes making up a node group stores a replica of the fragment for which that node group is responsible. Currently, a single cluster can support up to 48 data nodes total.

It is possible for more than one node to co-exist on a single machine. (In fact, it is even possible to set up a complete cluster on one machine, although one would almost certainly *not* want to do this in a production environment.) It may be helpful to remember that, when working with MySQL Cluster, the term *host* refers to a physical component of the cluster whereas a *node* is a logical or func-

tional component (that is, a process).

Note Regarding Terms. In volder versions of the MySQL Cluster documentation, data nodes were sometimes referred to as “database nodes”. The term “storage nodes” has also been used. In addition, SQL nodes were sometimes known as “client nodes”. This older terminology has been deprecated to minimize confusion, and for this reason should be avoided. They are also often referred to as “API nodes” — an SQL node is actually an API node that provides an SQL interface to the cluster.

- **Node group.** A set of data nodes. All data nodes in a node group contain the same data (fragments), and all nodes in a single group should reside on different hosts. It is possible to control which nodes belong to which node groups.

For more information, see [Section 16.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#).

- **Node failure.** MySQL Cluster is not solely dependent upon the functioning of any single node making up the cluster; the cluster can continue to run if one or more nodes fail. The precise number of node failures that a given cluster can tolerate depends upon the number of nodes and the cluster's configuration.
- **Node restart.** The process of restarting a failed cluster node.
- **Initial node restart.** The process of starting a cluster node with its filesystem removed. This is sometimes used in the course of software upgrades and in other special circumstances.
- **System crash (or system failure).** This can occur when so many cluster nodes have failed that the cluster's state can no longer be guaranteed.
- **System restart.** The process of restarting the cluster and reinitializing its state from disk logs and checkpoints. This is required after either a planned or an unplanned shutdown of the cluster.
- **Fragment.** A portion of a database table; in the [NDB](#) storage engine, a table is broken up into and stored as a number of fragments. A fragment is sometimes also called a “partition”; however, “fragment” is the preferred term. Tables are fragmented in MySQL Cluster in order to facilitate load balancing between machines and nodes.
- **Replica.** Under the [NDB](#) storage engine, each table fragment has number of replicas stored on other data nodes in order to provide redundancy. Currently, there may be up four replicas per fragment.
- **Transporter.** A protocol providing data transfer between nodes. MySQL Cluster currently supports four different types of transporter connections:
 - **TCP/IP.** This is, of course, the familiar network protocol that underlies HTTP, FTP (and so on) on the Internet. TCP/IP can be used for both local and remote connections.
 - **SCI.** Scalable Coherent Interface is a high-speed protocol used in building multiprocessor systems and parallel-processing applications. Use of SCI with MySQL Cluster requires specialized hardware, as discussed in [Section 16.12.1, “Configuring MySQL Cluster to use SCI Sockets”](#). For a basic introduction to SCI, see [this essay at dolphinics.com](#).
 - **SHM.** Unix-style shared memory segments. Where supported, SHM is used automatically to connect nodes running on the same host. The [Unix man page for shmop\(2\)](#) is a good place to begin obtaining additional information about this topic.

Note

The cluster transporter is internal to the cluster. Applications using MySQL Cluster communicate with SQL nodes just as they do with any other version of MySQL Server (via TCP/IP, or through the use of Unix socket files or Windows named pipes). Queries can be sent and results retrieved using the standard MySQL client APIs.

- **NDB.** This stands for Network Database, and refers to the storage engine used to enable MySQL Cluster. The [NDB](#) storage engine supports all the usual MySQL data types and SQL statements, and is ACID-compliant. This engine also provides full support for transactions (commits and rollbacks).
- **Shared-nothing architecture.** The ideal architecture for a MySQL Cluster. In a true shared-nothing setup, each node runs on a separate host. The advantage such an arrangement is that there no single host or node can act as single point of failure or as a performance bottle neck for the system as a whole.
- **In-memory storage.** All data stored in each data node is kept in memory on the node's host computer. For each data node in the cluster, you must have available an amount of RAM equal to the size of the database times the number of replicas, divided by the number of data nodes. Thus, if the database takes up 1GB of memory, and you want to set up the cluster with four replicas and eight data nodes, a minimum of 500MB memory will be required per node. Note that this is in addition to any requirements for the operating system and any other applications that might be running on the host.

In MySQL 5.1, it is also possible to create *Disk Data* tables where non-indexed columns are stored on disk, thus reducing the memory footprint required by the cluster. Note that indexes and indexed column data are still stored in RAM. See [Section 16.11, “MySQL Cluster Disk Data Tables”](#).

- **Table.** As is usual in the context of a relational database, the term “table” denotes a set of identically structured records. In MySQL Cluster, a database table is stored in a data node as a set of fragments, each of which is replicated on additional data nodes. The set of data nodes replicating the same fragment or set of fragments is referred to as a *node group*.
- **Cluster programs.** These are command-line programs used in running, configuring, and administering MySQL Cluster. They include both server daemons:

- `ndbd`:

The data node daemon (runs a data node process)

- `ndb_mgmd`:

The management server daemon (runs a management server process)

and client programs:

- `ndb_mgm`:

The management client (provides an interface for executing management commands)

- `ndb_waiter`:

Used to verify status of all nodes in a cluster

- `ndb_restore`:

Restores cluster data from backup

For more about these programs and their uses, see [Section 16.6, “Process Management in MySQL Cluster”](#).

- **Event log.** MySQL Cluster logs events by category (startup, shutdown, errors, checkpoints, and so on), priority, and severity. A complete listing of all reportable events may be found in [Section 16.7.3, “Event Reports Generated in MySQL Cluster”](#). Event logs are of two types:
 - **Cluster log.** Keeps a record of all desired reportable events for the cluster as a whole.
 - **Node log.** A separate log which is also kept for each individual node.

Under normal circumstances, it is necessary and sufficient to keep and examine only the cluster log. The node logs need be consulted only for application development and debugging purposes.

Chapter 17. Partitioning

This chapter discusses *user-defined partitioning*, as implemented in MySQL 5.1.

An introduction to partitioning and partitioning concepts may be found in [Section 17.1, “Overview of Partitioning in MySQL”](#).

MySQL supports several types of partitioning, which are discussed in [Section 17.2, “Partition Types”](#), as well as subpartitioning, which is described in [Section 17.2.5, “Subpartitioning”](#).

Methods of adding, removing, and altering partitions in existing partitioned tables are covered in [Section 17.3, “Partition Management”](#).

Table maintenance commands for use with partitioned tables are discussed in [Section 17.3.3, “Maintenance of Partitions”](#).

Important

Partitioned tables created with MySQL versions prior to 5.1.6 cannot be read by a 5.1.6 or later MySQL Server. In addition, the `INFORMATION_SCHEMA.TABLES` table cannot be used if such tables are present on a 5.1.6 server. Beginning with MySQL 5.1.7, a suitable warning message is generated instead, to alert the user that incompatible partitioned tables have been found by the server.

If you are using partitioned tables which were created in MySQL 5.1.5 or earlier, be sure to see [Section C.1.24, “Changes in release 5.1.6 \(01 February 2006\)”](#) for more information and suggested workarounds *before* upgrading to MySQL 5.1.6 or later.

The partitioning implementation in MySQL 5.1 is still undergoing development. For known issues with MySQL partitioning, see [Section 17.5, “Restrictions and Limitations on Partitioning”](#), where we have noted these.

You may also find the following resources to be useful when working with partitioned tables.

Additional Resources. Other sources of information about user-defined partitioning in MySQL include the following:

- [MySQL Partitioning Forum](#)

This is the official discussion forum for those interested in or experimenting with MySQL Partitioning technology. It features announcements and updates from MySQL developers and others. It is monitored by members of the Partitioning Development and Documentation Teams.

- [Mikael Ronström's Blog](#)

MySQL Partitioning Architect and Lead Developer Mikael Ronström frequently posts articles here concerning his work with MySQL Partitioning and MySQL Cluster.

- [PlanetMySQL](#)

A MySQL news site featuring MySQL-related blogs, which should be of interest to anyone using my MySQL. We encourage you to check here for links to blogs kept by those working with MySQL Partitioning, or to have your own blog added to those covered.

MySQL 5.1 binaries are now available from <http://dev.mysql.com/downloads/mysql/5.1.html>. However, for the latest partitioning bug-fixes and feature additions, you can obtain the source from our BitKeeper repository. To enable partitioning, you need to compile the server using the `--with-partition` option. For more information about building MySQL, see [Section 2.9, “MySQL Installation Using a Source Distribution”](#). If you have problems compiling a partitioning-enabled MySQL 5.1 build, check the [MySQL Partitioning Forum](#) and ask for assistance there if you don't find a solution to your problem already posted.

17.1. Overview of Partitioning in MySQL

This section provides a conceptual overview of partitioning in MySQL 5.1.

For information on partitioning restrictions and feature limitations, see [Section 17.5, “Restrictions and Limitations on Partitioning”](#).

The SQL standard does not provide much in the way of guidance regarding the physical aspects of data storage. The SQL language itself is intended to work independently of any data structures or media underlying the schemas, tables, rows, or columns with which it works. Nonetheless, most advanced database management systems have evolved some means of determining the physical location to be

used for storing specific pieces of data in terms of the filesystem, hardware or even both. In MySQL, the **InnoDB** storage engine has long supported the notion of a tablespace, and the MySQL Server, even prior to the introduction of partitioning, could be configured to employ different physical directories for storing different databases (see [Section 6.6.1, “Using Symbolic Links”](#), for an explanation of how this is done).

Partitioning takes this notion a step further, by allowing you to distribute portions of individual tables across a filesystem according to rules which you can set largely as needed. In effect, different portions of a table are stored as separate tables in different locations. The user-selected rule by which the division of data is accomplished is known as a *partitioning function*, which in MySQL can be the modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function. The function is selected according to the partitioning type specified by the user, and takes as its parameter the value of a user-supplied expression. This expression can be either an integer column value, or a function acting on one or more column values and returning an integer. The value of this expression is passed to the partitioning function, which returns an integer value representing the number of the partition in which that particular record should be stored. This function must be non-constant and non-random. It may not contain any queries, but may use an SQL expression that is valid in MySQL, as long as that expression returns either **NULL** or an integer *interval* such that

```
-MAXVALUE < interval < MAXVALUE
```

(**MAXVALUE** is used to represent the greatest possible positive integer.) There are some additional restrictions on partitioning functions; see [Section 17.5, “Restrictions and Limitations on Partitioning”](#), for more information about these.

Examples of partitioning functions can be found in the discussions of partitioning types later in this chapter (see [Section 17.2, “Partition Types”](#)), as well as in the partitioning syntax descriptions given in [Section 12.1.10, “CREATE TABLE Syntax”](#).

This is known as *horizontal partitioning* — that is, different rows of a table may be assigned to different physical partitions. MySQL 5.1 does not support *vertical partitioning*, in which different columns of a table are assigned to different physical partitions. There are not at this time any plans to introduce vertical partitioning into MySQL 5.1.

Partitioning support is included in the **-max** releases of MySQL 5.1 (that is, 5.1 **-max** binaries are built using **--with-partition**). If the MySQL binary is built with partitioning support, nothing further needs to be done in order to enable it (for example, no special entries are required in your **my.cnf** file). You can determine whether your MySQL server supports partitioning by means of a **SHOW VARIABLES** command such as this one:

```
mysql> SHOW VARIABLES LIKE '%partition%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_partitioning | YES   |
+-----+-----+
1 row in set (0.00 sec)
```

If you do not see the **have_partitioning** variable with the value **YES** listed as shown above in the output of an appropriate **SHOW VARIABLES**, then your version of MySQL does not support partitioning.

Prior to MySQL 5.1.6, this variable was named **have_partition_engine**. ([Bug#16718](#))

For creating partitioned tables, you can use most storage engines that are supported by your MySQL server; the MySQL partitioning engine runs in a separate layer and can interact with any of these. In MySQL 5.1, all partitions of the same partitioned table must use the same storage engine; for example, you cannot use **MyISAM** for one partition and **InnoDB** for another. However, there is nothing preventing you from using different storage engines for different partitioned tables on the same MySQL server or even in the same database.

Note

MySQL partitioning cannot be used with the **MERGE** or **CSV** storage engines. Beginning with MySQL 5.1.15, **FEDERATED** tables also cannot be partitioned ([Bug#22451](#)). Prior to MySQL 5.1.6, it was also not feasible to create a partitioned table using the **BLACKHOLE** storage engine ([Bug#14524](#)). Partitioning by **KEY** is supported for use with the **NDB-Cluster** storage engine, but other types of user-defined partitioning are not supported for Cluster tables in MySQL 5.1.

To employ a particular storage engine for a partitioned table, it is necessary only to use the **[STORAGE] ENGINE** option just as you would for a non-partitioned table. However, you should keep in mind that **[STORAGE] ENGINE** (and other table options) need to be listed *before* any partitioning options are used in a **CREATE TABLE** statement. This example shows how to create a table that is partitioned by hash into 6 partitions and which uses the **InnoDB** storage engine:

```
CREATE TABLE ti (id INT, amount DECIMAL(7,2), tr_date DATE)
ENGINE=INNODB
PARTITION BY HASH( MONTH(tr_date) )
PARTITIONS 6;
```


(Note that each **PARTITION** clause can include a **[STORAGE] ENGINE** option, but in MySQL 5.1 this has no effect.)

Important

Partitioning applies to all data and indexes of a table; you cannot partition only the data and not the indexes, or *vice versa*, nor can you partition only a portion of the table.

Data and indexes for each partition can be assigned to a specific directory using the **DATA DIRECTORY** and **INDEX DIRECTORY** options for the **PARTITION** clause of the **CREATE TABLE** statement used to create the partitioned table.

Note

Prior to MySQL 5.1.18, these options were permitted even when the **NO_DIR_IN_CREATE** server SQL mode was in effect. ([Bug#24633](#))

Important

On Windows, you must use the **/** character and not the **** character when specifying paths for **DATA DIRECTORY** and **INDEX DIRECTORY**, as shown in this example:

```
CREATE TABLE sales1 (
  id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  sales_date DATE NOT NULL DEFAULT '0000-00-00'
) ENGINE=InnoDB DEFAULT CHARSET=latin1
PARTITION BY RANGE(id)
(
  PARTITION p0 VALUES LESS THAN (5000)
    DATA DIRECTORY='d:/s1/data/'
    INDEX DIRECTORY = 'd:/s1/idx/',

  PARTITION p1 VALUES LESS THAN (10000)
    DATA DIRECTORY='e:/s1/data/'
    INDEX DIRECTORY = 'e:/s1/idx/',

  PARTITION p1 VALUES LESS THAN MAXVALUE
    DATA DIRECTORY='f:/s1/data/'
    INDEX DIRECTORY = 'f:/s1/idx/'
);
```

In addition, **MAX_ROWS** and **MIN_ROWS** can be used to determine the maximum and minimum numbers of rows, respectively, that can be stored in each partition. See [Section 17.3, “Partition Management”](#), for more information on these options.

Some of the advantages of partitioning include:

- Being able to store more data in one table than can be held on a single disk or filesystem partition.
- Data that loses its usefulness can often be easily removed from the table by dropping the partition containing only that data. Conversely, the process of adding new data can in some cases be greatly facilitated by adding a new partition specifically for that data.
- Some queries can be greatly optimized in virtue of the fact that data satisfying a given **WHERE** clause can be stored only on one or more partitions, thereby excluding any remaining partitions from the search. Because partitions can be altered after a partitioned table has been created, you can reorganize your data to enhance frequent queries that may not have been so when the partitioning scheme was first set up. This capability, sometimes referred to as *partition pruning*, was implemented in MySQL 5.1.6. For additional information, see [Section 17.4, “Partition Pruning”](#).

Other benefits usually associated with partitioning include those in the following list. These features are not currently implemented in MySQL Partitioning, but are high on our list of priorities.

- Queries involving aggregate functions such as **SUM()** and **COUNT()** can easily be parallelized. A simple example of such a query might be **SELECT salesperson_id, COUNT(orders) as order_total FROM sales GROUP BY salesperson_id;**. By “parallelized,” we mean that the query can be run simultaneously on each partition, and the final result obtained merely by summing the results obtained for all partitions.
- Achieving greater query throughput in virtue of spreading data seeks over multiple disks.

Be sure to check this section and chapter frequently for updates as Partitioning development continues.

17.2. Partition Types

This section discusses the types of partitioning which are available in MySQL 5.1. These include:

- **RANGE partitioning:** Assigns rows to partitions based on column values falling within a given range. See [Section 17.2.1, “RANGE Partitioning”](#).
- **LIST partitioning:** Similar to partitioning by range, except that the partition is selected based on columns matching one of a set of discrete values. See [Section 17.2.2, “LIST Partitioning”](#).
- **HASH partitioning:** A partition is selected based on the value returned by a user-defined expression that operates on column values in rows to be inserted into the table. The function may consist of any expression valid in MySQL that yields a non-negative integer value. See [Section 17.2.3, “HASH Partitioning”](#).
- **KEY partitioning:** Similar to partitioning by hash, except that only one or more columns to be evaluated are supplied, and the MySQL server provides its own hashing function. These columns can contain other than integer values, since the hashing function supplied by MySQL guarantees an integer result regardless of the column data type. See [Section 17.2.4, “KEY Partitioning”](#).

A very common use of database partitioning is to segregate data by date. Some database systems support explicit date partitioning, which MySQL does not implement in 5.1. However, it is not difficult in MySQL to create partitioning schemes based on [DATE](#), [TIME](#), or [DATETIME](#) columns, or based on expressions making use of such columns.

When partitioning by [KEY](#) or [LINEAR KEY](#), you can use a [DATE](#), [TIME](#), or [DATETIME](#) column as the partitioning column without performing any modification of the column value. For example, this table creation statement is perfectly valid in MySQL:

```
CREATE TABLE members (
  firstname VARCHAR(25) NOT NULL,
  lastname VARCHAR(25) NOT NULL,
  username VARCHAR(16) NOT NULL,
  email VARCHAR(35),
  joined DATE NOT NULL
)
PARTITION BY KEY(joined)
PARTITIONS 6;
```

MySQL's other partitioning types, however, require a partitioning expression that yields an integer value or [NULL](#). If you wish to use date-based partitioning by [RANGE](#), [LIST](#), [HASH](#), or [LINEAR HASH](#), you can simply employ a function that operates on a [DATE](#), [TIME](#), or [DATETIME](#) column and returns such a value, as shown here:

```
CREATE TABLE members (
  firstname VARCHAR(25) NOT NULL,
  lastname VARCHAR(25) NOT NULL,
  username VARCHAR(16) NOT NULL,
  email VARCHAR(35),
  joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
  PARTITION p0 VALUES LESS THAN (1960),
  PARTITION p1 VALUES LESS THAN (1970),
  PARTITION p2 VALUES LESS THAN (1980),
  PARTITION p3 VALUES LESS THAN (1990),
  PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Additional examples of partitioning using dates may be found here:

- [Section 17.2.1, “RANGE Partitioning”](#)
- [Section 17.2.3, “HASH Partitioning”](#)
- [Section 17.2.3.1, “LINEAR HASH Partitioning”](#)

For more complex examples of date-based partitioning, see:

- [Section 17.4, “Partition Pruning”](#)

- [Section 17.2.5, “Subpartitioning”](#)

MySQL partitioning is optimized for use with the `TO_DAYS()` and `YEAR()` functions. However, you can use other date and time functions that return an integer or `NULL`, such as `WEEKDAY()`, `DAYOFYEAR()`, or `MONTH()`. See [Section 11.6, “Date and Time Functions”](#), for more information about such functions.

It is important to remember — regardless of the type of partitioning that you use — that partitions are always numbered automatically and in sequence when created, starting with `0`. When a new row is inserted into a partitioned table, it is these partition numbers that are used in identifying the correct partition. For example, if your table uses 4 partitions, these partitions are numbered `0`, `1`, `2`, and `3`. For the `RANGE` and `LIST` partitioning types, it is necessary to ensure that there is a partition defined for each partition number. For `HASH` partitioning, the user function employed must return an integer value greater than `0`. For `KEY` partitioning, this issue is taken care of automatically by the hashing function which the MySQL server employs internally.

Names of partitions generally follow the rules governing other MySQL identifiers, such as those for tables and databases. However, you should note that partition names are not case-sensitive. For example, the following `CREATE TABLE` statement fails as shown:

```
mysql> CREATE TABLE t2 (val INT)
-> PARTITION BY LIST(val)(
->     PARTITION mypart VALUES IN (1,3,5),
->     PARTITION MyPart VALUES IN (2,4,6)
-> );
ERROR 1488 (HY000): Duplicate partition name mypart
```

Failure occurs because MySQL sees no difference between the partition names `mypart` and `MyPart`.

When you specify the number of partitions for the table, this must be expressed as a positive, non-zero integer literal with no leading zeroes, and may not be an expression such as `0.8E+01` or `6-2`, even if it evaluates as an integer. (Beginning with MySQL 5.1.12, decimal fractions are no longer truncated, but instead are disallowed entirely.)

In the sections that follow, we do not necessarily provide all possible forms for the syntax that can be used for creating each partition type; this information may be found in [Section 12.1.10, “CREATE TABLE Syntax”](#).

17.2.1. RANGE Partitioning

A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range. Ranges should be contiguous but not overlapping, and are defined using the `VALUES LESS THAN` operator. For the next few examples, suppose that you are creating a table such as the following to hold personnel records for a chain of 20 video stores, numbered 1 through 20:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
);
```

This table can be partitioned by range in a number of ways, depending on your needs. One way would be to use the `store_id` column. For instance, you might decide to partition the table 4 ways by adding a `PARTITION BY RANGE` clause as shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
  PARTITION p0 VALUES LESS THAN (6),
  PARTITION p1 VALUES LESS THAN (11),
  PARTITION p2 VALUES LESS THAN (16),
  PARTITION p3 VALUES LESS THAN (21)
);
```

In this partitioning scheme, all rows corresponding to employees working at stores 1 through 5 are stored in partition `p0`, to those employed at stores 6 through 10 are stored in partition `p1`, and so on. Note that each partition is defined in order, from lowest to highest.

This is a requirement of the **PARTITION BY RANGE** syntax; you can think of it as being analogous to a **switch ... case** in C or Java in this regard.

It is easy to determine that a new row containing the data (72, 'Michael', 'Widenius', '1998-06-25', NULL, 13) is inserted into partition **p2**, but what happens when your chain adds a 21st store? Under this scheme, there is no rule that covers a row whose **store_id** is greater than 20, so an error results because the server does not know where to place it. You can keep this from occurring by using a “catchall” **VALUES LESS THAN** clause in the **CREATE TABLE** statement that provides for all values greater than highest value explicitly named:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
  PARTITION p0 VALUES LESS THAN (6),
  PARTITION p1 VALUES LESS THAN (11),
  PARTITION p2 VALUES LESS THAN (16),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

MAXVALUE represents the greatest possible integer value. Now, any rows whose **store_id** column value is greater than or equal to 16 (the highest value defined) are stored in partition **p3**. At some point in the future — when the number of stores has increased to 25, 30, or more — you can use an **ALTER TABLE** statement to add new partitions for stores 21-25, 26-30, and so on (see [Section 17.3, “Partition Management”](#), for details of how to do this).

In much the same fashion, you could partition the table based on employee job codes — that is, based on ranges of **job_code** column values. For example — assuming that two-digit job codes are used for regular (in-store) workers, three-digit codes are used for office and support personnel, and four-digit codes are used for management positions — you could create the partitioned table using the following:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
)
PARTITION BY RANGE (job_code) (
  PARTITION p0 VALUES LESS THAN (100),
  PARTITION p1 VALUES LESS THAN (1000),
  PARTITION p2 VALUES LESS THAN (10000)
);
```

In this instance, all rows relating to in-store workers would be stored in partition **p0**, those relating to office and support staff in **p1**, and those relating to managers in partition **p2**.

It is also possible to use an expression in **VALUES LESS THAN** clauses. However, MySQL must be able to evaluate the expression's return value as part of a **LESS THAN (<)** comparison.

Rather than splitting up the table data according to store number, you can use an expression based on one of the two **DATE** columns instead. For example, let us suppose that you wish to partition based on the year that each employee left the company; that is, the value of **YEAR(separated)**. An example of a **CREATE TABLE** statement that implements such a partitioning scheme is shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY RANGE ( YEAR(separated) ) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1996),
  PARTITION p2 VALUES LESS THAN (2001),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

In this scheme, for all employees who left before 1991, the rows are stored in partition `p0`; for those who left in the years 1991 through 1995, in `p1`; for those who left in the years 1996 through 2000, in `p2`; and for any workers who left after the year 2000, in `p3`.

Range partitioning is particularly useful when:

- You want or need to delete “old” data. If you are using the partitioning scheme shown immediately above, you can simply use `ALTER TABLE employees DROP PARTITION p0`; to delete all rows relating to employees who stopped working for the firm prior to 1991. (See [Section 12.1.4, “ALTER TABLE Syntax”](#), and [Section 17.3, “Partition Management”](#), for more information.) For a table with a great many rows, this can be much more efficient than running a `DELETE` query such as `DELETE FROM employees WHERE YEAR(separated) <= 1990`;
- You want to use a column containing date or time values, or containing values arising from some other series.
- You frequently run queries that depend directly on the column used for partitioning the table. For example, when executing a query such as `SELECT COUNT(*) FROM employees WHERE YEAR(separated) = 2000 GROUP BY store_id`;, MySQL can quickly determine that only partition `p2` needs to be scanned because the remaining partitions cannot contain any records satisfying the `WHERE` clause. See [Section 17.4, “Partition Pruning”](#), for more information about how this is accomplished.

17.2.2. LIST Partitioning

List partitioning in MySQL is similar to range partitioning in many ways. As in partitioning by `RANGE`, each partition must be explicitly defined. The chief difference is that, in list partitioning, each partition is defined and selected based on the membership of a column value in one of a set of value lists, rather than in one of a set of contiguous ranges of values. This is done by using `PARTITION BY LIST(expr)` where `expr` is a column value or an expression based on a column value and returning an integer value, and then defining each partition by means of a `VALUES IN (value_list)`, where `value_list` is a comma-separated list of integers.

Note

In MySQL 5.1, it is possible to match against only a list of integers (and possibly `NULL` — see [Section 17.2.6, “How MySQL Partitioning Handles NULL Values”](#)) when partitioning by `LIST`.

Unlike the case with partitions defined by range, list partitions do not need to be declared in any particular order. For more detailed syntactical information, see [Section 12.1.10, “CREATE TABLE Syntax”](#).

For the examples that follow, we assume that the basic definition of the table to be partitioned is provided by the `CREATE TABLE` statement shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
);
```

(This is the same table used as a basis for the examples in [Section 17.2.1, “RANGE Partitioning”](#).)

Suppose that there are 20 video stores distributed among 4 franchises as shown in the following table:

Region	Store ID Numbers
North	3, 5, 6, 9, 17
East	1, 2, 10, 11, 19, 20
West	4, 12, 13, 14, 18
Central	7, 8, 15, 16

To partition this table in such a way that rows for stores belonging to the same region are stored in the same partition, you could use the `CREATE TABLE` statement shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
```

```

    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LIST(store_id) (
    PARTITION pNorth VALUES IN (3,5,6,9,17),
    PARTITION pEast VALUES IN (1,2,10,11,19,20),
    PARTITION pWest VALUES IN (4,12,13,14,18),
    PARTITION pCentral VALUES IN (7,8,15,16)
);

```

This makes it easy to add or drop employee records relating to specific regions to or from the table. For instance, suppose that all stores in the West region are sold to another company. All rows relating to employees working at stores in that region can be deleted with the query `ALTER TABLE employees DROP PARTITION pWest;`, which can be executed much more efficiently than the equivalent `DELETE` statement `DELETE FROM employees WHERE store_id IN (4,12,13,14,18);`.

As with [RANGE](#) partitioning, it is possible to combine [LIST](#) partitioning with partitioning by hash or key to produce a composite partitioning (subpartitioning). See [Section 17.2.5, “Subpartitioning”](#).

17.2.3. HASH Partitioning

Partitioning by [HASH](#) is used primarily to ensure an even distribution of data among a predetermined number of partitions. With range or list partitioning, you must specify explicitly into which partition a given column value or set of column values is to be stored; with hash partitioning, MySQL takes care of this for you, and you need only specify a column value or expression based on a column value to be hashed and the number of partitions into which the partitioned table is to be divided.

To partition a table using [HASH](#) partitioning, it is necessary to append to the `CREATE TABLE` statement a `PARTITION BY HASH (expr)` clause, where *expr* is an expression that returns an integer. This can simply be the name of a column whose type is one of MySQL's integer types. In addition, you will most likely want to follow this with a `PARTITIONS num` clause, where *num* is a non-negative integer representing the number of partitions into which the table is to be divided.

For example, the following statement creates a table that uses hashing on the `store_id` column and is divided into 4 partitions:

```

CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;

```

If you do not include a `PARTITIONS` clause, the number of partitions defaults to `1`.

Using the `PARTITIONS` keyword without a number following it results in a syntax error.

You can also use an SQL expression that returns an integer for *expr*. For instance, you might want to partition based on the year in which an employee was hired. This can be done as shown here:

```

CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH( YEAR(hired) )
PARTITIONS 4;

```

You may use any function or other expression for *expr* that is valid in MySQL, so long as it returns a non-constant, non-random integer value. (In other words, it should be varying but deterministic.) However, you should keep in mind that this expression is evaluated each time a row is inserted or updated (or possibly deleted); this means that very complex expressions may give rise to performance issues, particularly when performing operations (such as batch inserts) that affect a great many rows at one time.

The most efficient hashing function is one which operates upon a single table column and whose value increases or decreases consistently with the column value, as this allows for “pruning” on ranges of partitions. That is, the more closely that the expression varies with the value of the column on which it is based, the more efficiently MySQL can use the expression for hash partitioning.

For example, where `date_col` is a column of type `DATE`, then the expression `TO_DAYS(date_col)` is said to vary directly with the value of `date_col`, because for every change in the value of `date_col`, the value of the expression changes in a consistent manner. The variance of the expression `YEAR(date_col)` with respect to `date_col` is not quite as direct as that of `TO_DAYS(date_col)`, because not every possible change in `date_col` produces an equivalent change in `YEAR(date_col)`. Even so, `YEAR(date_col)` is a good candidate for a hashing function, because it varies directly with a portion of `date_col` and there is no possible change in `date_col` that produces a disproportionate change in `YEAR(date_col)`.

By way of contrast, suppose that you have a column named `int_col` whose type is `INT`. Now consider the expression `POW(5-int_col,3) + 6`. This would be a poor choice for a hashing function because a change in the value of `int_col` is not guaranteed to produce a proportional change in the value of the expression. Changing the value of `int_col` by a given amount can produce widely different changes in the value of the expression. For example, changing `int_col` from 5 to 6 produces a change of -1 in the value of the expression, but changing the value of `int_col` from 6 to 7 produces a change of -7 in the expression value.

In other words, the more closely the graph of the column value *versus* the value of the expression follows a straight line as traced by the equation $y=nx$ where n is some nonzero constant, the better the expression is suited to hashing. This has to do with the fact that the more nonlinear an expression is, the more uneven the distribution of data among the partitions it tends to produce.

In theory, pruning is also possible for expressions involving more than one column value, but determining which of such expressions are suitable can be quite difficult and time-consuming. For this reason, the use of hashing expressions involving multiple columns is not particularly recommended.

When `PARTITION BY HASH` is used, MySQL determines which partition of `num` partitions to use based on the modulus of the result of the user function. In other words, for an expression `expr`, the partition in which the record is stored is partition number N , where $N = \text{MOD}(\text{expr}, \text{num})$. For example, suppose table `t1` is defined as follows, so that it has 4 partitions:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY HASH( YEAR(col3) )
PARTITIONS 4;
```

If you insert a record into `t1` whose `col3` value is `'2005-09-15'`, then the partition in which it is stored is determined as follows:

```
MOD(YEAR('2005-09-01'),4)
= MOD(2005,4)
= 1
```

MySQL 5.1 also supports a variant of `HASH` partitioning known as *linear hashing* which employs a more complex algorithm for determining the placement of new rows inserted into the partitioned table. See [Section 17.2.3.1, “LINEAR HASH Partitioning”](#), for a description of this algorithm.

The user function is evaluated each time a record is inserted or updated. It may also — depending on the circumstances — be evaluated when records are deleted.

Note

If a table to be partitioned has a `UNIQUE` key, then any columns supplied as arguments to the `HASH` user function or to the `KEY's column_list` must be part of that key.

17.2.3.1. LINEAR HASH Partitioning

MySQL also supports linear hashing, which differs from regular hashing in that linear hashing utilizes a linear powers-of-two algorithm whereas regular hashing employs the modulus of the hashing function's value.

Syntactically, the only difference between linear-hash partitioning and regular hashing is the addition of the `LINEAR` keyword in the `PARTITION BY` clause, as shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
```

```
PARTITION BY LINEAR HASH( YEAR(hired) )
PARTITIONS 4;
```

Given an expression *expr*, the partition in which the record is stored when linear hashing is used is partition number *N* from among *num* partitions, where *N* is derived according to the following algorithm:

1. Find the next power of 2 greater than *num*. We call this value *V*; it can be calculated as:

```
V = POWER(2, CEILING(LOG(2, num)))
```

(For example, suppose that *num* is 13. Then $\text{LOG}(2, 13)$ is 3.7004397181411. $\text{CEILING}(3.7004397181411)$ is 4, and $V = \text{POWER}(2, 4)$, which is 16.)

2. Set $N = F(\text{column\_list}) \& (V - 1)$.
3. While $N \geq \text{num}$:
 - Set $V = \text{CEIL}(V / 2)$
 - Set $N = N \& (V - 1)$

For example, suppose that the table *t1*, using linear hash partitioning and having 6 partitions, is created using this statement:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY LINEAR HASH( YEAR(col3) )
PARTITIONS 6;
```

Now assume that you want to insert two records into *t1* having the *col3* column values '2003-04-14' and '1998-10-19'. The partition number for the first of these is determined as follows:

```
V = POWER(2, CEILING( LOG(2, 7) )) = 8
N = YEAR('2003-04-14') & (8 - 1)
  = 2003 & 7
  = 3
(3 >= 6 is FALSE: record stored in partition #3)
```

The number of the partition where the second record is stored is calculated as shown here:

```
V = 8
N = YEAR('1998-10-19') & (8-1)
  = 1998 & 7
  = 6
(6 >= 6 is TRUE: additional step required)
N = 6 & CEILING(5 / 2)
  = 6 & 3
  = 2
(2 >= 6 is FALSE: record stored in partition #2)
```

The advantage in partitioning by linear hash is that the adding, dropping, merging, and splitting of partitions is made much faster, which can be beneficial when dealing with tables containing extremely large amounts (terabytes) of data. The disadvantage is that data is less likely to be evenly distributed between partitions as compared with the distribution obtained using regular hash partitioning.

17.2.4. KEY Partitioning

Partitioning by key is similar to partitioning by hash, except that where hash partitioning employs a user-defined expression, the hashing function for key partitioning is supplied by the MySQL server. MySQL Cluster uses `MD5()` for this purpose; for tables using other storage engines, the server employs its own internal hashing function which is based on the same algorithm as `PASSWORD()`.

The syntax rules for `CREATE TABLE ... PARTITION BY KEY` are similar to those for creating a table that is partitioned by hash. The major differences are that:

- **KEY** is used rather than **HASH**.
- **KEY** takes only a list of one or more column names. Beginning with MySQL 5.1.5, the column or columns used as the partitioning key must comprise part or all of the table's primary key, if the table has one.

Beginning with MySQL 5.1.6, **KEY** takes a list of zero or more column names. Where no column name is specified as the partitioning key, the table's primary key is used, if there is one. For example, the following **CREATE TABLE** statement is valid in MySQL 5.1.6 or later:

```
CREATE TABLE k1 (
  id INT NOT NULL PRIMARY KEY,
  name VARCHAR(20)
)
PARTITION BY KEY()
PARTITIONS 2;
```

If there is no primary key but there is a unique key, then the unique key is used for the partitioning key:

```
CREATE TABLE k1 (
  id INT NOT NULL,
  name VARCHAR(20),
  UNIQUE KEY (id)
)
PARTITION BY KEY()
PARTITIONS 2;
```

However, if the unique key column were not defined as **NOT NULL**, then the previous statement would fail.

In both of these cases, the partitioning key is the **id** column, even though it is not shown in the output of **SHOW CREATE TABLE** or in the **PARTITION_EXPRESSION** column of the **INFORMATION_SCHEMA.PARTITIONS** table.

Unlike the case with other partitioning types, columns used for partitioning by **KEY** are not restricted to integer or **NULL** values. For example, the following **CREATE TABLE** statement is valid:

```
CREATE TABLE tm1 (
  s1 CHAR(32) PRIMARY KEY
)
PARTITION BY KEY(s1)
PARTITIONS 10;
```

The preceding statement would *not* be valid, were a different partitioning type to be specified.

Note

In this case, simply using **PARTITION BY KEY()** would also be valid and have the same effect as **PARTITION BY KEY(s1)**, since **s1** is the table's primary key.

For additional information about this issue, see [Section 17.5, “Restrictions and Limitations on Partitioning”](#).

Note

Also beginning with MySQL 5.1.6, tables using the **NDB Cluster** storage engine are implicitly partitioned by **KEY**, again using the table's primary key as the partitioning key. In the event that the Cluster table has no explicit primary key, the “hidden” primary key generated by the **NDB** storage engine for each Cluster table is used as the partitioning key.

Important

For a key-partitioned table using any MySQL storage engine other than **NDB Cluster**, you cannot execute an **ALTER TABLE DROP PRIMARY KEY**, as doing so generates the error **ERROR 1466 (HY000): FIELD IN LIST OF FIELDS FOR PARTITION FUNCTION NOT FOUND IN TABLE**. This is not an issue for MySQL Cluster tables which are partitioned by **KEY**; in such cases, the table is reorganized using the “hidden” primary key as the table's new partitioning key. See [Chapter 16, MySQL Cluster](#).

It is also possible to partition a table by linear key. Here is a simple example:

```
CREATE TABLE tk (
  col1 INT NOT NULL,
  col2 CHAR(5),
```

```
col3 DATE
)
PARTITION BY LINEAR KEY (col1)
PARTITIONS 3;
```

Using **LINEAR** has the same effect on **KEY** partitioning as it does on **HASH** partitioning, with the partition number being derived using a powers-of-two algorithm rather than modulo arithmetic. See [Section 17.2.3.1, “LINEAR HASH Partitioning”](#), for a description of this algorithm and its implications.

17.2.5. Subpartitioning

Subpartitioning — also known as *composite partitioning* — is the further division of each partition in a partitioned table. For example, consider the following **CREATE TABLE** statement:

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) )
SUBPARTITIONS 2 (
    PARTITION p0 VALUES LESS THAN (1990),
    PARTITION p1 VALUES LESS THAN (2000),
    PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

Table **ts** has 3 **RANGE** partitions. Each of these partitions — **p0**, **p1**, and **p2** — is further divided into 2 subpartitions. In effect, the entire table is divided into $3 * 2 = 6$ partitions. However, due to the action of the **PARTITION BY RANGE** clause, the first 2 of these store only those records with a value less than 1990 in the **purchased** column.

In MySQL 5.1, it is possible to subpartition tables that are partitioned by **RANGE** or **LIST**. Subpartitions may use either **HASH** or **KEY** partitioning. This is also known as *composite partitioning*.

It is also possible to define subpartitions explicitly using **SUBPARTITION** clauses to specify options for individual subpartitions. For example, a more verbose fashion of creating the same table **ts** as shown in the previous example would be:

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
    PARTITION p0 VALUES LESS THAN (1990) (
        SUBPARTITION s0,
        SUBPARTITION s1
    ),
    PARTITION p1 VALUES LESS THAN (2000) (
        SUBPARTITION s2,
        SUBPARTITION s3
    ),
    PARTITION p2 VALUES LESS THAN MAXVALUE (
        SUBPARTITION s4,
        SUBPARTITION s5
    )
);
```

Some syntactical items of note:

- Each partition must have the same number of subpartitions.
- If you explicitly define any subpartitions using **SUBPARTITION** on any partition of a partitioned table, you must define them all. In other words, the following statement will fail:

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
    PARTITION p0 VALUES LESS THAN (1990) (
        SUBPARTITION s0,
        SUBPARTITION s1
    ),
    PARTITION p1 VALUES LESS THAN (2000),
    PARTITION p2 VALUES LESS THAN MAXVALUE (
        SUBPARTITION s2,
        SUBPARTITION s3
    )
);
```

This statement would still fail even if it included a **SUBPARTITIONS 2** clause.

- Each **SUBPARTITION** clause must include (at a minimum) a name for the subpartition. Otherwise, you may set any desired option for the subpartition or allow it to assume its default setting for that option.
- In MySQL 5.1.7 and earlier, names of subpartitions must be unique within each partition, but do not have to be unique within the table as a whole. Beginning with MySQL 5.1.8, subpartition names must be unique across the entire table. For example, the following **CREATE TABLE** statement is valid in MySQL 5.1.8 and later:

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0,
    SUBPARTITION s1
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s2,
    SUBPARTITION s3
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s4,
    SUBPARTITION s5
  )
);
```

(The previous statement is also valid for versions of MySQL prior to 5.1.8.)

Subpartitions can be used with especially large tables to distribute data and indexes across many disks. Suppose that you have 6 disks mounted as **/disk0**, **/disk1**, **/disk2**, and so on. Now consider the following example:

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0
      DATA DIRECTORY = '/disk0/data'
      INDEX DIRECTORY = '/disk0/idx',
    SUBPARTITION s1
      DATA DIRECTORY = '/disk1/data'
      INDEX DIRECTORY = '/disk1/idx'
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s2
      DATA DIRECTORY = '/disk2/data'
      INDEX DIRECTORY = '/disk2/idx',
    SUBPARTITION s3
      DATA DIRECTORY = '/disk3/data'
      INDEX DIRECTORY = '/disk3/idx'
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s4
      DATA DIRECTORY = '/disk4/data'
      INDEX DIRECTORY = '/disk4/idx',
    SUBPARTITION s5
      DATA DIRECTORY = '/disk5/data'
      INDEX DIRECTORY = '/disk5/idx'
  )
);
```

In this case, a separate disk is used for the data and for the indexes of each **RANGE**. Many other variations are possible; another example might be:

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE(YEAR(purchased))
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0a
      DATA DIRECTORY = '/disk0'
      INDEX DIRECTORY = '/disk1',
    SUBPARTITION s0b
      DATA DIRECTORY = '/disk2'
      INDEX DIRECTORY = '/disk3'
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s1a
      DATA DIRECTORY = '/disk4/data'
      INDEX DIRECTORY = '/disk4/idx',
    SUBPARTITION s1b
      DATA DIRECTORY = '/disk5/data'
      INDEX DIRECTORY = '/disk5/idx'
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s2a,
```

```

        SUBPARTITION s2b
    );

```

Here, the storage is as follows:

- Rows with **purchased** dates from before 1990 take up a vast amount of space, so are split up 4 ways, with a separate disk dedicated to the data and to the indexes for each of the two subpartitions (**s0a** and **s0b**) making up partition **p0**. In other words:
 - The data for subpartition **s0a** is stored on **/disk0**.
 - The indexes for subpartition **s0a** are stored on **/disk1**.
 - The data for subpartition **s0b** is stored on **/disk2**.
 - The indexes for subpartition **s0b** are stored on **/disk3**.
- Rows containing dates ranging from 1990 to 1999 (partition **p1**) do not require as much room as those from before 1990. These are split between 2 disks (**/disk4** and **/disk5**) rather than 4 disks as with the legacy records stored in **p0**:
 - Data and indexes belonging to **p1**'s first subpartition (**s1a**) are stored on **/disk4** — the data in **/disk4/data**, and the indexes in **/disk4/idx**.
 - Data and indexes belonging to **p1**'s second subpartition (**s1b**) are stored on **/disk5** — the data in **/disk5/data**, and the indexes in **/disk5/idx**.
- Rows reflecting dates from the year 2000 to the present (partition **p2**) do not take up as much space as required by either of the two previous ranges. Currently, it is sufficient to store all of these in the default location.

In future, when the number of purchases for the decade beginning with the year 2000 grows to a point where the default location no longer provides sufficient space, the corresponding rows can be moved using an **ALTER TABLE ... REORGANIZE PARTITION** statement. See [Section 17.3, “Partition Management”](#), for an explanation of how this can be done.

Beginning with MySQL 5.1.18, the **DATA DIRECTORY** and **INDEX DIRECTORY** options are disallowed when the **NO_DIR_IN_CREATE** server SQL mode is in effect. This is true for partitions and subpartitions.

17.2.6. How MySQL Partitioning Handles **NULL** Values

Partitioning in MySQL does nothing to disallow **NULL** as the value of a partitioning expression, whether it is a column value or the value of a user-supplied expression. Even though it is permitted to use **NULL** as the value of an expression that must otherwise yield an integer, it is important to keep in mind that **NULL** is not a number. Beginning version 5.1.8, MySQL Partitioning treats **NULL** as being less than any non-**NULL** value, just as **ORDER BY** does.

Because of this, this treatment of **NULL** varies between partitioning of different types, and may produce behavior which you do not expect if you are not prepared for it. This being the case, we discuss in this section how each MySQL partitioning types handles **NULL** values when determining the partition in which a row should be stored, and provide examples for each.

If you insert a row into a table partitioned by **RANGE** such that the column value used to determine the partition is **NULL**, the row is inserted into the lowest partition. For example, consider these two tables, created and populated as follows:

```

mysql> CREATE TABLE t1 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY RANGE(c1) (
->   PARTITION p0 VALUES LESS THAN (0),
->   PARTITION p1 VALUES LESS THAN (10),
->   PARTITION p2 VALUES LESS THAN MAXVALUE
-> );
Query OK, 0 rows affected (0.09 sec)

mysql> CREATE TABLE t1 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY RANGE(c1) (
->   PARTITION p0 VALUES LESS THAN (-5),
->   PARTITION p1 VALUES LESS THAN (0),
->   PARTITION p1 VALUES LESS THAN (10),

```

```

-> PARTITION p2 VALUES LESS THAN MAXVALUE
-> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t1 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM t1;
+-----+-----+
| id    | name  |
+-----+-----+
| NULL  | mothra |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
+-----+-----+
| id    | name  |
+-----+-----+
| NULL  | mothra |
+-----+-----+
1 row in set (0.00 sec)

```

You can see which partitions the rows are stored in by inspecting the filesystem and comparing the sizes of the `.MYD` files corresponding to the partitions:

```

/var/lib/mysql/test> ls -l *.MYD
-rw-rw---- 1 mysql mysql 20 2006-03-10 03:27 t1#P#p0.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 03:17 t1#P#p1.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 03:17 t1#P#p2.MYD
-rw-rw---- 1 mysql mysql 20 2006-03-10 03:27 t2#P#p0.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 03:17 t2#P#p1.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 03:17 t2#P#p2.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 03:17 t2#P#p3.MYD

```

Partition files are named according to the format `table_name#P#partition_name.extension`, so that `t1#P#p0.MYD` is the file in which data belonging to partition `p0` of table `t1` is stored.

Note

Prior to MySQL 5.1.5, these files would have been named `t1_p0.MYD` and `t2_p0.MYD`, respectively. See [Section C.1.24, “Changes in release 5.1.6 \(01 February 2006\)”](#) and [Bug#13437](#) for information regarding how this change impacts upgrades.

You can also demonstrate that these rows were stored in the lowest partition of the each table by dropping these partitions, and then re-running the `SELECT` statements:

```

mysql> ALTER TABLE t1 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> ALTER TABLE t2 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT * FROM t1;
Empty set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)

```

(For more information on `ALTER TABLE ... DROP PARTITION`, see [Section 12.1.4, “ALTER TABLE Syntax”](#).)

Such treatment also holds true for partitioning expressions that use SQL functions. Suppose that we have a table such as this one:

```

CREATE TABLE tndate (
  id INT,
  dt DATE
)
PARTITION BY RANGE( YEAR(dt) ) (
  PARTITION p0 VALUES LESS THAN (1990),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN MAXVALUE
);

```

As with other MySQL functions, `YEAR(NULL)` returns `NULL`. A row with a `dt` column value of `NULL` is treated as though the partitioning expression evaluated to a value less than any other value, and so is inserted into partition `p0`.

A table that is partitioned by **LIST** admits **NULL** values if and only if one of its partitions is defined using that value-list that contains **NULL**. The converse of this is that a table partitioned by **LIST** which does not explicitly use **NULL** in a value list rejects rows resulting in a **NULL** value for the partitioning expression, as shown in this example:

```
mysql> CREATE TABLE ts1 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY LIST(c1) (
->   PARTITION p0 VALUES IN (0, 3, 6),
->   PARTITION p1 VALUES IN (1, 4, 7),
->   PARTITION p2 VALUES IN (2, 5, 8)
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts1 VALUES (9, 'mothra');
ERROR 1504 (HY000): TABLE HAS NO PARTITION FOR VALUE 9

mysql> INSERT INTO ts1 VALUES (NULL, 'mothra');
ERROR 1504 (HY000): TABLE HAS NO PARTITION FOR VALUE NULL
```

Only rows having a **c1** value between 0 and 8 inclusive can be inserted into **ts1**. **NULL** falls outside this range, just like the number 9. We can create tables **ts2** and **ts3** having value lists containing **NULL**, as shown here:

```
mysql> CREATE TABLE ts2 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY LIST(c1) (
->   PARTITION p0 VALUES IN (0, 3, 6),
->   PARTITION p1 VALUES IN (1, 4, 7),
->   PARTITION p2 VALUES IN (2, 5, 8),
->   PARTITION p3 VALUES IN (NULL)
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE ts3 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY LIST(c1) (
->   PARTITION p0 VALUES IN (0, 3, 6),
->   PARTITION p1 VALUES IN (1, 4, 7, NULL),
->   PARTITION p2 VALUES IN (2, 5, 8)
-> );
Query OK, 0 rows affected (0.01 sec)
```

When defining value lists for partitioning, you can treat **NULL** just as you would any other value, and so **VALUES IN (NULL)** and **VALUES IN (1, 4, 7, NULL)** are both valid (as are **VALUES IN (1, NULL, 4, 7)**, **VALUES IN (NULL, 1, 4, 7)**, and so on). You can insert a row having **NULL** for column **c1** into each of the tables **ts2** and **ts3**:

```
mysql> INSERT INTO ts2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ts3 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
```

By inspecting the filesystem, you can verify that the first of these statements inserted a new row into partition **p3** of table **ts2**, and that the second statement inserted a new row into partition **p1** of table **ts3**:

```
/var/lib/mysql/test> ls -l ts2*.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 10:35 ts2#P#p0.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 10:35 ts2#P#p1.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 10:35 ts2#P#p2.MYD
-rw-rw---- 1 mysql mysql 20 2006-03-10 10:35 ts2#P#p3.MYD

/var/lib/mysql/test> ls -l ts3*.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 10:36 ts3#P#p0.MYD
-rw-rw---- 1 mysql mysql 20 2006-03-10 10:36 ts3#P#p1.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 10:36 ts3#P#p2.MYD
```

As in earlier examples, we assume the use of the **bash** shell on a Unix operating system for listing files; use whatever your platform provides in this regard. For example, if you are using a DOS shell on a Windows operating system, the equivalent for the last listing might be obtained by running the command **dir ts3*.MYD** in the directory **C:\Program Files\MySQL\MySQL Server 5.1\data\test**.

As shown earlier in this section, you can also verify which partitions were used for storing the values by deleting them and then per-

forming a `SELECT`.

`NULL` is handled somewhat differently for tables partitioned by `HASH` or `KEY`. In these cases, any partition expression that yields a `NULL` value is treated as though its return value were zero. We can verify this behavior by examining the effects on the filesystem of creating a table partitioned by `HASH` and populating it with a record containing appropriate values. Suppose that you have a table `th`, created in the `test` database, using this statement:

```
mysql> CREATE TABLE th (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY HASH(c1)
-> PARTITIONS 2;
Query OK, 0 rows affected (0.00 sec)
```

Assuming an RPM installation of MySQL on Linux, this statement creates two `.MYD` files in `/var/lib/mysql/test`, which can be viewed in the `bash` shell as follows:

```
/var/lib/mysql/test> ls th*.MYD -l
-rw-rw---- 1 mysql mysql 0 2005-11-04 18:41 th#P#p0.MYD
-rw-rw---- 1 mysql mysql 0 2005-11-04 18:41 th#P#p1.MYD
```

Note that the size of each file is 0 bytes. Now insert a row into `th` whose `c1` column value is `NULL`, and verify that this row was inserted:

```
mysql> INSERT INTO th VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM th;
+-----+-----+
| c1    | c2    |
+-----+-----+
| NULL  | mothra |
+-----+-----+
1 row in set (0.01 sec)
```

Recall that for any integer `N`, the value of `NULL MOD N` is always `NULL`. For tables that are partitioned by `HASH` or `KEY`, this result is treated for determining the correct partition as `0`. Returning to the system shell (still assuming `bash` for this purpose), we can see that the value was inserted into the first partition (named `p0` by default) by listing the data files once again:

```
var/lib/mysql/test> ls *.MYD -l
-rw-rw---- 1 mysql mysql 20 2005-11-04 18:44 th#P#p0.MYD
-rw-rw---- 1 mysql mysql 0 2005-11-04 18:41 th#P#p1.MYD
```

You can see that the `INSERT` statement modified only the file `th#P#p0.MYD` (increasing its size on disk), without affecting the other data file.

Important

Prior to MySQL 5.1.8, `RANGE` partitioning treated a partitioning expression value of `NULL` as a zero with respect to determining placement (the only way to circumvent this was to design tables so as not to allow nulls, usually by declaring columns `NOT NULL`). If you have a `RANGE` partitioning scheme that depends on this earlier behavior, you will need to re-implement it when upgrading to MySQL 5.1.8 or later.

17.3. Partition Management

MySQL 5.1 provides a number of ways to modify partitioned tables. It is possible to add, drop, redefine, merge, or split existing partitions. All of these actions can be carried out using the partitioning extensions to the `ALTER TABLE` command (see [Section 12.1.4, “ALTER TABLE Syntax”](#), for syntax definitions). There are also ways to obtain information about partitioned tables and partitions. We discuss these topics in the sections that follow.

- For information about partition management in tables partitioned by `RANGE` or `LIST`, see [Section 17.3.1, “Management of RANGE and LIST Partitions”](#).
- For a discussion of managing `HASH` and `KEY` partitions, see [Section 17.3.2, “Management of HASH and KEY Partitions”](#).
- See [Section 17.3.4, “Obtaining Information About Partitions”](#), for a discussion of mechanisms provided in MySQL 5.1 for obtaining

information about partitioned tables and partitions.

- For a discussion of performing maintenance operations on partitions, see [Section 17.3.3, “Maintenance of Partitions”](#).

Note

In MySQL 5.1, all partitions of a partitioned table must have the same number of subpartitions, and it is not possible to change the subpartitioning once the table has been created.

The statement `ALTER TABLE ... PARTITION BY ...` is available and is functional beginning with MySQL 5.1.6; previously in MySQL 5.1, this was accepted as valid syntax, but the statement did nothing.

To change a table's partitioning scheme, it is necessary only to use the `ALTER TABLE` command with a *partition_options* clause. This clause has the same syntax as that as used with `CREATE TABLE` for creating a partitioned table, and always begins with the keywords `PARTITION BY`. For example, suppose that you have a table partitioned by range using the following `CREATE TABLE` statement:

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
PARTITION BY RANGE( YEAR(purchased) ) (
    PARTITION p0 VALUES LESS THAN (1990),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (2000),
    PARTITION p3 VALUES LESS THAN (2005)
);
```

To repartition this table so that it is partitioned by key into two partitions using the `id` column value as the basis for the key, you can use this statement:

```
ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;
```

This has the same effect on the structure of the table as dropping the table and re-creating it using `CREATE TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;`.

Important

In MySQL 5.1.7 and earlier MySQL 5.1 releases, `ALTER TABLE ... ENGINE = ...` removed all partitioning from the affected table. Beginning with MySQL 5.1.8, this statement changes only the storage engine used by the table, and leaves the table's partitioning scheme intact. As of MySQL 5.1.8, use `ALTER TABLE ... REMOVE PARTITIONING` to remove a table's partitioning. See [Section 12.1.4, “ALTER TABLE Syntax”](#).

17.3.1. Management of RANGE and LIST Partitions

Range and list partitions are very similar with regard to how the adding and dropping of partitions are handled. For this reason we discuss the management of both sorts of partitioning in this section. For information about working with tables that are partitioned by hash or key, see [Section 17.3.2, “Management of HASH and KEY Partitions”](#). Dropping a `RANGE` or `LIST` partition is more straightforward than adding one, so we discuss this first.

Dropping a partition from a table that is partitioned by either `RANGE` or by `LIST` can be accomplished using the `ALTER TABLE` statement with a `DROP PARTITION` clause. Here is a very basic example, which supposes that you have already created a table which is partitioned by range and then populated with 10 records using the following `CREATE TABLE` and `INSERT` statements:

```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
-> PARTITION BY RANGE( YEAR(purchased) ) (
-> PARTITION p0 VALUES LESS THAN (1990),
-> PARTITION p1 VALUES LESS THAN (1995),
-> PARTITION p2 VALUES LESS THAN (2000),
-> PARTITION p3 VALUES LESS THAN (2005)
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO tr VALUES
-> (1, 'desk organiser', '2003-10-15'),
-> (2, 'CD player', '1993-11-05'),
-> (3, 'TV set', '1996-03-10'),
-> (4, 'bookcase', '1982-01-10'),
-> (5, 'exercise bike', '2004-05-09'),
-> (6, 'sofa', '1987-06-05'),
-> (7, 'popcorn maker', '2001-11-22'),
-> (8, 'aquarium', '1992-08-04'),
-> (9, 'study desk', '1984-09-16'),
```



```
-> (10, 'lava lamp', '1998-12-25');
Query OK, 10 rows affected (0.01 sec)
```

You can see which items should have been inserted into partition `p2` as shown here:

```
mysql> SELECT * FROM tr
-> WHERE purchased BETWEEN '1995-01-01' AND '1999-12-31';
+-----+-----+-----+
| id | name | purchased |
+-----+-----+-----+
| 3 | TV set | 1996-03-10 |
| 10 | lava lamp | 1998-12-25 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

To drop the partition named `p2`, execute the following command:

```
mysql> ALTER TABLE tr DROP PARTITION p2;
Query OK, 0 rows affected (0.03 sec)
```

Note: In MySQL 5.1, the `NDBCLUSTER` storage engine does not support `ALTER TABLE ... DROP PARTITION`. It does, however, support the other partitioning-related extensions to `ALTER TABLE` that are described in this chapter.

It is very important to remember that, *when you drop a partition, you also delete all the data that was stored in that partition*. You can see that this is the case by re-running the previous `SELECT` query:

```
mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '1999-12-31';
Empty set (0.00 sec)
```

Because of this, the requirement was added in MySQL 5.1.10 that you have the `DROP` privilege for a table before you can execute `ALTER TABLE ... DROP PARTITION` on that table.

If you wish to drop all data from all partitions while preserving the table definition and its partitioning scheme, use the `TRUNCATE TABLE` command. (See [Section 12.2.9, “TRUNCATE Syntax”](#).)

If you intend to change the partitioning of a table *without* losing data, use `ALTER TABLE ... REORGANIZE PARTITION` instead. See below or in [Section 12.1.4, “ALTER TABLE Syntax”](#), for information about `REORGANIZE PARTITION`.

If you now execute a `SHOW CREATE TABLE` command, you can see how the partitioning makeup of the table has been changed:

```
mysql> SHOW CREATE TABLE tr\G
***** 1. row *****
Table: tr
Create Table: CREATE TABLE `tr` (
  `id` int(11) default NULL,
  `name` varchar(50) default NULL,
  `purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE ( YEAR(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
1 row in set (0.01 sec)
```

When you insert new rows into the changed table with `purchased` column values between `'1995-01-01'` and `'2004-12-31'` inclusive, those rows will be stored in partition `p3`. You can verify this as follows:

```
mysql> INSERT INTO tr VALUES (11, 'pencil holder', '1995-07-12');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '2004-12-31';
+-----+-----+-----+
| id | name | purchased |
+-----+-----+-----+
| 11 | pencil holder | 1995-07-12 |
| 1 | desk organiser | 2003-10-15 |
| 5 | exercise bike | 2004-05-09 |
| 7 | popcorn maker | 2001-11-22 |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> ALTER TABLE tr DROP PARTITION p3;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '2004-12-31';
Empty set (0.00 sec)
```

Note that the number of rows dropped from the table as a result of `ALTER TABLE ... DROP PARTITION` is not reported by the server as it would be by the equivalent `DELETE` query.

Dropping `LIST` partitions uses exactly the same `ALTER TABLE ... DROP PARTITION` syntax as used for dropping `RANGE` partitions. However, there is one important difference in the effect this has on your use of the table afterward: You can no longer insert into the table any rows having any of the values that were included in the value list defining the deleted partition. (See [Section 17.2.2](#), “`LIST` Partitioning”, for an example.)

To add a new range or list partition to a previously partitioned table, use the `ALTER TABLE ... ADD PARTITION` statement. For tables which are partitioned by `RANGE`, this can be used to add a new range to the end of the list of existing partitions. For example, suppose that you have a partitioned table containing membership data for your organisation, which is defined as follows:

```
CREATE TABLE members (
  id INT,
  fname VARCHAR(25),
  lname VARCHAR(25),
  dob DATE
)
PARTITION BY RANGE( YEAR(dob) ) (
  PARTITION p0 VALUES LESS THAN (1970),
  PARTITION p1 VALUES LESS THAN (1980),
  PARTITION p2 VALUES LESS THAN (1990)
);
```

Suppose further that the minimum age for members is 16. As the calendar approaches the end of 2005, you realize that you will soon be admitting members who were born in 1990 (and later in years to come). You can modify the `members` table to accommodate new members born in the years 1990-1999 as shown here:

```
ALTER TABLE ADD PARTITION (PARTITION p3 VALUES LESS THAN (2000));
```

Important

With tables that are partitioned by range, you can use `ADD PARTITION` to add new partitions to the high end of the partitions list only. Trying to add a new partition in this manner between or before existing partitions will result in an error as shown here:

```
mysql> ALTER TABLE members
> ADD PARTITION (
> PARTITION p3 VALUES LESS THAN (1960));
ERROR 1463 (HY000): VALUES LESS THAN value must be strictly »
increasing for each partition
```

In a similar fashion, you can add new partitions to a table that is partitioned by `LIST`. For example, given a table defined like so:

```
CREATE TABLE tt (
  id INT,
  data INT
)
PARTITION BY LIST(data) (
  PARTITION p0 VALUES IN (5, 10, 15),
  PARTITION p1 VALUES IN (6, 12, 18)
);
```

You can add a new partition in which to store rows having the `data` column values `7`, `14`, and `21` as shown:

```
ALTER TABLE tt ADD PARTITION (PARTITION p2 VALUES IN (7, 14, 21));
```

Note that you *cannot* add a new `LIST` partition encompassing any values that are already included in the value list of an existing partition. If you attempt to do so, an error will result:

```
mysql> ALTER TABLE tt ADD PARTITION
> (PARTITION np VALUES IN (4, 8, 12));
ERROR 1465 (HY000): Multiple definition of same constant »
in list partitioning
```

Because any rows with the `data` column value `12` have already been assigned to partition `p1`, you cannot create a new partition on table `tt` that includes `12` in its value list. To accomplish this, you could drop `p1`, and add `np` and then a new `p1` with a modified definition. However, as discussed earlier, this would result in the loss of all data stored in `p1` — and it is often the case that this is not what you really want to do. Another solution might appear to be to make a copy of the table with the new partitioning and to copy the data into it using `CREATE TABLE ... SELECT ...`, then drop the old table and rename the new one, but this could be very time-consuming when dealing with a large amounts of data. This also might not be feasible in situations where high availability is a requirement.

Beginning with MySQL 5.1.6, you can add multiple partitions in a single `ALTER TABLE ... ADD PARTITION` statement as shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  hired DATE NOT NULL
)
PARTITION BY RANGE( YEAR(hired) ) (
  PARTITION p1 VALUES LESS THAN (1991),
  PARTITION p2 VALUES LESS THAN (1996),
  PARTITION p3 VALUES LESS THAN (2001),
  PARTITION p4 VALUES LESS THAN (2005)
);

ALTER TABLE employees ADD PARTITION (
  PARTITION p5 VALUES LESS THAN (2010),
  PARTITION p6 VALUES LESS THAN MAXVALUE
);
```

Fortunately, MySQL's partitioning implementation provides ways to redefine partitions without losing data. Let us look first at a couple of simple examples involving `RANGE` partitioning. Recall the `members` table which is now defined as shown here:

```
mysql> SHOW CREATE TABLE members\G
***** 1. row *****
Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) default NULL,
  `fname` varchar(25) default NULL,
  `lname` varchar(25) default NULL,
  `dob` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE ( YEAR(dob) ) (
  PARTITION p0 VALUES LESS THAN (1970) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1980) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2000) ENGINE = MyISAM
)
```

Suppose that you would like to move all rows representing members born before 1960 into a separate partition. As we have already seen, this cannot be done using `ALTER TABLE ... ADD PARTITION`. However, you can use another partition-related extension to `ALTER TABLE` in order to accomplish this:

```
ALTER TABLE members REORGANIZE PARTITION p0 INTO (
  PARTITION s0 VALUES LESS THAN (1960),
  PARTITION s1 VALUES LESS THAN (1970)
);
```

In effect, this command splits partition `p0` into two new partitions `s0` and `s1`. It also moves the data that was stored in `p0` into the new partitions according to the rules embodied in the two `PARTITION ... VALUES ...` clauses, so that `s0` contains only those records for which `YEAR(dob)` is less than 1960 and `s1` contains those rows in which `YEAR(dob)` is greater than or equal to 1960 but less than 1970.

A `REORGANIZE PARTITION` clause may also be used for merging adjacent partitions. You can return the `members` table to its previous partitioning as shown here:

```
ALTER TABLE members REORGANIZE PARTITION s0,s1 INTO (
  PARTITION p0 VALUES LESS THAN (1970)
);
```

No data is lost in splitting or merging partitions using `REORGANIZE PARTITION`. In executing the above statement, MySQL moves all of the records that were stored in partitions `s0` and `s1` into partition `p0`.

The general syntax for `REORGANIZE PARTITION` is:

```
ALTER TABLE tbl_name
  REORGANIZE PARTITION partition_list
  INTO (partition_definitions);
```

Here, *tbl_name* is the name of the partitioned table, and *partition_list* is a comma-separated list of names of one or more existing partitions to be changed. *partition_definitions* is a comma-separated list of new partition definitions, which follow the same rules as for the *partition_definitions* list used in [CREATE TABLE](#) (see [Section 12.1.10, “CREATE TABLE Syntax”](#)). It should be noted that you are not limited to merging several partitions into one, or to splitting one partition into many, when using [REORGANIZE PARTITION](#). For example, you can reorganize all four partitions of the `members` table into two, as follows:

```
ALTER TABLE members REORGANIZE PARTITION p0,p1,p2,p3 INTO (
  PARTITION m0 VALUES LESS THAN (1980),
  PARTITION m1 VALUES LESS THAN (2000)
);
```

You can also use [REORGANIZE PARTITION](#) with tables that are partitioned by [LIST](#). Let us return to the problem of adding a new partition to the list-partitioned `tt` table and failing because the new partition had a value that was already present in the value-list of one of the existing partitions. We can handle this by adding a partition that contains only non-conflicting values, and then reorganizing the new partition and the existing one so that the value which was stored in the existing one is now moved to the new one:

```
ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8));
ALTER TABLE tt REORGANIZE PARTITION p1,np INTO (
  PARTITION p1 VALUES IN (6, 18),
  PARTITION np VALUES in (4, 8, 12)
);
```

Here are some key points to keep in mind when using [ALTER TABLE ... REORGANIZE PARTITION](#) to repartition tables that are partitioned by [RANGE](#) or [LIST](#):

- The [PARTITION](#) clauses used to determine the new partitioning scheme are subject to the same rules as those used with a [CREATE TABLE](#) statement.

Most importantly, you should remember that the new partitioning scheme cannot have any overlapping ranges (applies to tables partitioned by [RANGE](#)) or sets of values (when reorganizing tables partitioned by [LIST](#)).

Note

Prior to MySQL 5.1.4, you could not reuse the names of existing partitions in the `INTO` clause, even when those partitions were being dropped or redefined. See [Section C.1.26, “Changes in release 5.1.4 \(21 December 2005\)”](#), for more information.

- The combination of partitions in the *partition_definitions* list should account for the same range or set of values overall as the combined partitions named in the *partition_list*.

For instance, in the `members` table used as an example in this section, partitions `p1` and `p2` together cover the years 1980 through 1999. Therefore, any reorganization of these two partitions should cover the same range of years overall.

- For tables partitioned by [RANGE](#), you can reorganize only adjacent partitions; you cannot skip over range partitions.

For instance, you could not reorganize the `members` table used as an example in this section using a statement beginning with `ALTER TABLE members REORGANIZE PARTITION p0,p2 INTO ...` because `p0` covers the years prior to 1970 and `p2` the years from 1990 through 1999 inclusive, and thus the two are not adjacent partitions.

- You cannot use [REORGANIZE PARTITION](#) to change the table's partitioning type; that is, you cannot (for example) change [RANGE](#) partitions to [HASH](#) partitions or *vice versa*. You also cannot use this command to change the partitioning expression or column. To accomplish either of these tasks without dropping and re-creating the table, you can use [ALTER TABLE ... PARTITION BY ...](#). For example:

```
ALTER TABLE members
  PARTITION BY HASH( YEAR(dob) )
  PARTITIONS 8;
```

17.3.2. Management of [HASH](#) and [KEY](#) Partitions

Tables which are partitioned by hash or by key are very similar to one another with regard to making changes in a partitioning setup, and both differ in a number of ways from tables which have been partitioned by range or list. For that reason, this section addresses the modification of tables partitioned by hash or by key only. For a discussion of adding and dropping of partitions of tables that are partitioned by range or list, see [Section 17.3.1, “Management of RANGE and LIST Partitions”](#).

You cannot drop partitions from tables that are partitioned by [HASH](#) or [KEY](#) in the same way that you can from tables that are partitioned by [RANGE](#) or [LIST](#). However, you can merge [HASH](#) or [KEY](#) partitions using the [ALTER TABLE ... COALESCE PARTITION](#) command. For example, suppose that you have a table containing data about clients, which is divided into twelve partitions. The `clients` table is defined as shown here:

```
CREATE TABLE clients (
  id INT,
  fname VARCHAR(30),
  lname VARCHAR(30),
  signed DATE
)
PARTITION BY HASH( MONTH(signed) )
PARTITIONS 12;
```

To reduce the number of partitions from twelve to eight, execute the following [ALTER TABLE](#) command:

```
mysql> ALTER TABLE clients COALESCE PARTITION 4;
Query OK, 0 rows affected (0.02 sec)
```

[COALESCE](#) works equally well with tables that are partitioned by [HASH](#), [KEY](#), [LINEAR HASH](#), or [LINEAR KEY](#). Here is an example similar to the previous one, differing only in that the table is partitioned by [LINEAR KEY](#):

```
mysql> CREATE TABLE clients_lk (
->   id INT,
->   fname VARCHAR(30),
->   lname VARCHAR(30),
->   signed DATE
-> )
-> PARTITION BY LINEAR KEY(signed)
-> PARTITIONS 12;
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE clients_lk COALESCE PARTITION 4;
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Note that the number following [COALESCE PARTITION](#) is the number of partitions to merge into the remainder — in other words, it is the number of partitions to remove from the table.

If you attempt to remove more partitions than the table has, the result is an error like the one shown:

```
mysql> ALTER TABLE clients COALESCE PARTITION 18;
ERROR 1478 (HY000): Cannot remove all partitions, use DROP TABLE instead
```

To increase the number of partitions for the `clients` table from 12 to 18, use [ALTER TABLE ... ADD PARTITION](#) as shown here:

```
ALTER TABLE clients ADD PARTITION PARTITIONS 6;
```

17.3.3. Maintenance of Partitions

A number of partitioning maintenance tasks can be carried out in MySQL 5.1. MySQL does not support the commands [CHECK TABLE](#), [OPTIMIZE TABLE](#), [ANALYZE TABLE](#), or [REPAIR TABLE](#) for partitioned tables. Instead, you can use a number of extensions to [ALTER TABLE](#) which were implemented in MySQL 5.1.5. These can be used for performing operations of this type on one or more partitions directly, as described in the following list:

- **Rebuilding partitions.** Rebuilds the partition; this has the same effect as dropping all records stored in the partition, then reinserting them. This can be useful for purposes of defragmentation.

Example:

```
ALTER TABLE t1 REBUILD PARTITION p0, p1;
```

- **Optimizing partitions.** If you have deleted a large number of rows from a partition or if you have made many changes to a partitioned table with variable-length rows (that is, having [VARCHAR](#), [BLOB](#), or [TEXT](#) columns), you can use [ALTER TABLE ... OPTIMIZE PARTITION](#) to reclaim any unused space and to defragment the partition data file.

Example:

```
ALTER TABLE t1 OPTIMIZE PARTITION p0, p1;
```

Using [OPTIMIZE PARTITION](#) on a given partition is equivalent to running [CHECK PARTITION](#), [ANALYZE PARTITION](#), and [REPAIR PARTITION](#) on that partition.

- **Analyzing partitions.** This reads and stores the key distributions for partitions.

Example:

```
ALTER TABLE t1 ANALYZE PARTITION p3;
```

- **Repairing partitions.** This repairs corrupted partitions.

Example:

```
ALTER TABLE t1 REPAIR PARTITION p0,p1;
```

- **Checking partitions.** You can check partitions for errors in much the same way that you can use [CHECK TABLE](#) with non-partitioned tables.

Example:

```
ALTER TABLE trb3 CHECK PARTITION p1;
```

This command will tell you if the data or indexes in partition [p1](#) of table [t1](#) are corrupted. If this is the case, use [ALTER TABLE ... REPAIR PARTITION](#) to repair the partition.

You can also use the [mysqlcheck](#) or [myisamchk](#) utility to accomplish these tasks, operating on the separate [.MYI](#) files generated by partitioning a table. See [Section 7.12, “mysqlcheck — A Table Maintenance and Repair Program”](#).

17.3.4. Obtaining Information About Partitions

This section discusses obtaining information about existing partitions, which can be done in a number of ways. These include:

- Using the [SHOW CREATE TABLE](#) statement to view the partitioning clauses used in creating a partitioned table.
- Using the [SHOW TABLE STATUS](#) statement to determine whether a table is partitioned.
- Querying the [INFORMATION_SCHEMA.PARTITIONS](#) table.
- Using the statement [EXPLAIN PARTITIONS SELECT](#) to see which partitions are used by a given [SELECT](#).

As discussed elsewhere in this chapter, [SHOW CREATE TABLE](#) includes in its output the [PARTITION BY](#) clause used to create a partitioned table. For example:

```
mysql> SHOW CREATE TABLE trb3\G
***** 1. row *****
      Table: trb3
Create Table: CREATE TABLE `trb3` (
  `id` int(11) default NULL,
  `name` varchar(50) default NULL,
  `purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE (YEAR(purchased)) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (2000) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
```

1 row in set (0.00 sec)

Note

In early MySQL 5.1 releases, the `PARTITIONS` clause was not shown for tables partitioned by `HASH` or `KEY`. This issue was fixed in MySQL 5.1.6.

`SHOW TABLE STATUS` works with partitioned tables. Beginning with MySQL 5.1.9, its output is the same as that for non-partitioned tables, except that the `Create_options` column contains the string `partitioned`. In MySQL 5.1.8 and earlier, the `Engine` column always contained the value `PARTITION`; beginning with MySQL 5.1.9, this column contains the name of the storage engine used by all partitions of the table. (See [Section 12.5.4.28, “SHOW TABLE STATUS Syntax”](#), for more information about this command.)

You can also obtain information about partitions from `INFORMATION_SCHEMA`, which contains a `PARTITIONS` table. See [Section 23.19, “The INFORMATION_SCHEMA PARTITIONS Table”](#).

Beginning with MySQL 5.1.5, it is possible to determine which partitions of a partitioned table are involved in a given `SELECT` query using `EXPLAIN PARTITIONS`. The `PARTITIONS` keyword adds a `partitions` column to the output of `EXPLAIN` listing the partitions from which records would be matched by the query.

Suppose that you have a table `trb1` defined and populated as follows:

```
CREATE TABLE trb1 (id INT, name VARCHAR(50), purchased DATE)
PARTITION BY RANGE(id)
(
    PARTITION p0 VALUES LESS THAN (3),
    PARTITION p1 VALUES LESS THAN (7),
    PARTITION p2 VALUES LESS THAN (9),
    PARTITION p3 VALUES LESS THAN (11)
);

INSERT INTO trb1 VALUES
(1, 'desk organiser', '2003-10-15'),
(2, 'CD player', '1993-11-05'),
(3, 'TV set', '1996-03-10'),
(4, 'bookcase', '1982-01-10'),
(5, 'exercise bike', '2004-05-09'),
(6, 'sofa', '1987-06-05'),
(7, 'popcorn maker', '2001-11-22'),
(8, 'aquarium', '1992-08-04'),
(9, 'study desk', '1984-09-16'),
(10, 'lava lamp', '1998-12-25');
```

You can see which partitions are used in a query such as `SELECT * FROM trb1;`, as shown here:

```
mysql> EXPLAIN PARTITIONS SELECT * FROM trb1\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: trb1
    partitions: p0,p1,p2,p3
         type: ALL
possible_keys: NULL
         key: NULL
      key_len: NULL
         ref: NULL
        rows: 10
      Extra: Using filesort
```

In this case, all four partitions are searched. However, when a limiting condition making use of the partitioning key is added to the query, you can see that only those partitions containing matching values are scanned, as shown here:

```
mysql> EXPLAIN PARTITIONS SELECT * FROM trb1 WHERE id < 5\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: trb1
    partitions: p0,p1
         type: ALL
possible_keys: NULL
         key: NULL
      key_len: NULL
         ref: NULL
        rows: 10
      Extra: Using where
```

EXPLAIN PARTITIONS provides information about keys used and possible keys, just as with the standard **EXPLAIN SELECT** statement:

```
mysql> ALTER TABLE trb1 ADD PRIMARY KEY (id);
Query OK, 10 rows affected (0.03 sec)
Records: 10 Duplicates: 0 Warnings: 0

mysql> EXPLAIN PARTITIONS SELECT * FROM trb1 WHERE id < 5\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
         ref: NULL
        rows: 7
      Extra: Using where
```

You should take note of the following restrictions and limitations on **EXPLAIN PARTITIONS**:

- You cannot use the **PARTITIONS** and **EXTENDED** keywords together in the same **EXPLAIN . . . SELECT** statement. Attempting to do so produces a syntax error.
- If **EXPLAIN PARTITIONS** is used to examine a query against a non-partitioned table, no error is produced, but the value of the **partitions** column is always **NULL**.

See also [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#).

17.4. Partition Pruning

This section discusses *partition pruning*, an optimisation which was implemented for partitioned tables in MySQL 5.1.6. The core concept behind partition pruning is relatively simple, and can be described as “Do not scan partitions where there can be no matching values”. For example, suppose you have a partitioned table **t1** defined by this statement:

```
CREATE TABLE t1 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY RANGE( region_code ) (
  PARTITION p0 VALUES LESS THAN (64),
  PARTITION p1 VALUES LESS THAN (128),
  PARTITION p2 VALUES LESS THAN (192),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

Consider the case where you wish to obtain results from a query such as this one:

```
SELECT fname, lname, postcode, dob
FROM t1
WHERE region_code > 125 AND region_code < 130;
```

It is easy to see that none of the rows which ought to be returned will be in either of the partitions **p0** or **p3**; that is, we need to search only in partitions **p1** and **p2** to find matching rows. By doing so, it is possible to expend much more time and effort in finding matching rows than it is to scan all partitions in the table. This “cutting away” of unneeded partitions is known as *pruning*. When the optimizer can make use of partition pruning in performing a query, execution of the query can be an order of magnitude faster than the same query against a non-partitioned table containing the same column definitions and data.

The query optimizer can perform pruning whenever a **WHERE** condition can be reduced to either one of the following:

- *partition_column = constant*
- *partition_column IN (constant1, constant2, ..., constantN)*

In the first case, the optimizer simply evaluates the partitioning expression for the value given, determines which partition contains that value, and scans only this partition. In many cases, the equals sign can be replaced with another arithmetic comparison, including `<`, `>`, `<=`, `>=`, and `<>`. Some queries using `BETWEEN` in the `WHERE` clause can also take advantage of partition pruning. See the examples later in this section.

In the second case, the optimizer evaluates the partitioning expression for each value in the list, creates a list of matching partitions, and then scans only the partitions in this partition list.

Pruning can also be applied to short ranges, which the optimizer can convert into equivalent lists of values. For instance, in the previous example, the `WHERE` clause can be converted to `WHERE region_code IN (125, 126, 127, 128, 129, 130)`. Then the optimizer can determine that the first three values in the list are found in partition `p1`, the remaining three values in partition `p2`, and that the other partitions contain no relevant values and so do not need to be searched for matching rows.

This type of optimization can be applied whenever the partitioning expression consists of an equality or a range which can be reduced to a set of equalities, or when the partitioning expression represents an increasing or decreasing relationship. Pruning can also be applied for tables partitioned on a `DATE` or `DATETIME` column when the partitioning expression uses the `YEAR()` or `TO_DAYS()` function.

Note

We plan to add pruning support in a future MySQL release for additional functions that act on a `DATE` or `DATETIME` value, return an integer, and are increasing or decreasing.

For example, suppose that table `t2`, defined as shown here, is partitioned on a `DATE` column:

```
CREATE TABLE t2 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY RANGE( YEAR(dob) ) (
  PARTITION d0 VALUES LESS THAN (1970),
  PARTITION d1 VALUES LESS THAN (1975),
  PARTITION d2 VALUES LESS THAN (1980),
  PARTITION d3 VALUES LESS THAN (1985),
  PARTITION d4 VALUES LESS THAN (1990),
  PARTITION d5 VALUES LESS THAN (2000),
  PARTITION d6 VALUES LESS THAN (2005),
  PARTITION d7 VALUES LESS THAN MAXVALUE
);
```

The following queries on `t2` can make use of partition pruning:

```
SELECT * FROM t2 WHERE dob = '1982-06-23';
SELECT * FROM t2 WHERE dob BETWEEN '1991-02-15' AND '1997-04-25';
SELECT * FROM t2 WHERE YEAR(dob)
  IN (1979, 1980, 1983, 1985, 1986, 1988);
SELECT * FROM t2 WHERE dob >= '1984-06-21' AND dob <= '1999-06-21'
```

In the case of the last query, the optimizer can also act as follows:

1. Find the partition containing the low end of the range.
`YEAR( '1984-06-21' )` yields the value `1984`, which is found in partition `d3`.
2. Find the partition containing the high end of the range.
`YEAR( '1999-06-21' )` evaluates to `1999`, which is found in partition `d5`.
3. Scan only these two partitions and any partitions that may lie between them.

In this case, this means that only partitions `d3`, `d4`, and `d5` are scanned. The remaining partitions may be safely ignored (and are ignored).

So far, we have looked only at examples using `RANGE` partitioning, but pruning can be applied with other partitioning types as well.

Consider a table that is partitioned by `LIST`, where the partitioning expression is increasing or decreasing, such as the table `t3` shown

here. (In this example, we assume for the sake of brevity that the `region_code` column is limited to values between 1 and 10 inclusive.)

```
CREATE TABLE t3 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY LIST(region_code) (
  PARTITION r0 VALUES IN (1, 3),
  PARTITION r1 VALUES IN (2, 5, 8),
  PARTITION r2 VALUES IN (4, 9),
  PARTITION r3 VALUES IN (6, 7, 10)
);
```

For a query such as `SELECT * FROM t3 WHERE region_code BETWEEN 1 AND 3`, the optimizer determines in which partitions the values 1, 2, and 3 are found (`r0` and `r1`) and skips the remaining ones (`r2` and `r3`).

For tables that are partitioned by `HASH` or `KEY`, partition pruning is also possible in cases in which the `WHERE` clause uses a simple `=` relation against a column used in the partitioning expression. Consider a table created like this:

```
CREATE TABLE t4 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY KEY(region_code)
PARTITIONS 8;
```

Any query such as this one can be pruned:

```
SELECT * FROM t4 WHERE region_code = 7;
```

Pruning can also be employed for short ranges, because the optimizer can turn such conditions into `IN` relations. For example, using the same table `t4` as defined previously, queries such as these can be pruned:

```
SELECT * FROM t4 WHERE region_code > 2 AND region_code < 6;
SELECT * FROM t4 WHERE region_code BETWEEN 3 AND 5;
```

In both these cases, the `WHERE` clause is transformed by the optimizer into `WHERE region_code IN (3, 4, 5)`.

Important

This optimization is used only if the range size is smaller than the number of partitions. Consider this query:

```
SELECT * FROM t4 WHERE region_code BETWEEN 4 AND 8;
```

The range in the `WHERE` clause covers 5 values (4, 5, 6, 7, 8), but `t4` has only 4 partitions. This means that the previous query cannot be pruned.

Pruning can be used only on integer columns of tables partitioned by `HASH` or `KEY`. For example, this query on table `t4` cannot use pruning because `dob` is a `DATE` column:

```
SELECT * FROM t4 WHERE dob >= '2001-04-14' AND dob <= '2005-10-15';
```

However, if the table stores year values in an `INT` column, then a query having `WHERE year_col >= 2001 AND year_col <= 2005` can be pruned.

17.5. Restrictions and Limitations on Partitioning

This section discusses current restrictions and limitations on MySQL partitioning support, as listed here:

- **Prohibited constructs.** Beginning with MySQL 5.1.12, the following constructs are not permitted in partitioning expressions:
 - Stored functions, stored procedures, UDFs, or plugins.

- Declared variables or user variables.

For SQL functions which are not permitted in partitioning expressions, see [Section 17.5.3.2, “Functions Not Permitted in Partitioning Expressions”](#).

- **Arithmetic and logical operators.** Use of the arithmetic operators `+`, `-`, `*`, and `/` is permitted in partitioning expressions. However, the result must be an integer value or `NULL` (except in the case of `[LINEAR] KEY` partitioning, as discussed elsewhere in this chapter — see [Section 17.2, “Partition Types”](#), for more information).

Beginning with MySQL 5.1.12, the bit operators `|`, `&`, `^`, `<<`, `>>`, and `~` are not permitted in partitioning expressions.

- **Server SQL mode.** Tables employing user-defined partitioning do not preserve the SQL mode in effect at the time that they were created. As discussed in [Section 5.2.6, “SQL Modes”](#), the results of many MySQL functions and operators may change according to the server SQL mode. Therefore, a change in the SQL mode at any time after the creation of partitioned tables may lead to major changes in the behavior of such tables, and could easily lead to corruption or loss of data. For these reasons, *it is strongly recommended that you never change the server SQL mode after creating partitioned tables*.

- **Maximum number of partitions.** The maximum number of partitions possible for a given table is 1024. This includes subpartitions.

If, when creating tables with a very large number of partitions (but which is less than the maximum stated previously), you encounter an error message such as `GOT ERROR 24 FROM STORAGE ENGINE`, this means that you may need to increase the value of the `open_files_limit` system variable. See [Section B.1.2.17, “‘FILE’ NOT FOUND and Similar Errors”](#).

- **Foreign keys.** Partitioned tables do not support foreign keys. This includes partitioned tables employing the `InnoDB` storage engine.
- **FULLTEXT indexes.** Partitioned tables do not support `FULLTEXT` indexes. This includes partitioned tables employing the `MyISAM` storage engine.
- **GEOMETRY columns.** Partitioned tables do not support `GEOMETRY` columns.
- **Temporary tables.** As of MySQL 5.1.8, temporary tables cannot be partitioned. ([Bug#17497](#))
- **Log tables.** Beginning with MySQL 5.1.20, it is no longer possible to partition the log tables; beginning with that version, an `ALTER TABLE ... PARTITION BY ...` statement on such a table fails with an error. ([Bug#27816](#))
- **Data type of partitioning key.** A partitioning key must be either an integer column or an expression that resolves to an integer. The column or expression value may also be `NULL`. (See [Section 17.2.6, “How MySQL Partitioning Handles NULL Values”](#).)

The lone exception to this restriction occurs when partitioning by `[LINEAR] KEY` — where it is possible to use columns of other types as partitioning keys — because MySQL’s internal key-hashing functions produce the correct data type from these types. For example, the following `CREATE TABLE` statement is valid:

```
CREATE TABLE tkc (c1 CHAR)
PARTITION BY KEY(c1)
PARTITIONS 4;
```

This exception does *not* apply to `BLOB` or `TEXT` column types.

- **Subqueries.** A partitioning key may not be a subquery, even if that subquery resolves to an integer value or `NULL`.
- **Subpartitions.** Subpartitions are limited to `HASH` or `KEY` partitioning. `HASH` and `KEY` partitions cannot be subpartitioned.
- **LOAD INDEX INTO CACHE.** The `LOAD INDEX INTO CACHE` statement is not supported for partitioned tables. Attempting to load an index from a partitioned table into a key cache results in the error `THE STORAGE ENGINE FOR THE TABLE DOESN'T SUPPORT PRELOAD_KEYS`.

17.5.1. Partitioning Keys, Primary Keys, and Unique Keys

This section discusses the relationship of partitioning keys with primary keys and unique keys. The rule governing this relationship can be expressed as follows: All columns used in the partitioning expression for a partitioned table must be part of every unique key that the table may have.

In other words, *every unique key on the table must use every column in the table's partitioning expression*. (This also includes the table's primary key, since it is by definition a unique key. This particular case is discussed later in this section.) For example, each of the following table creation statements is invalid:

```
CREATE TABLE t1 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  UNIQUE KEY (col1, col2)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  UNIQUE KEY (col1),
  UNIQUE KEY (col3)
)
PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

In each case, the proposed table would have at least one unique key that does not include all columns used in the partitioning expression.

Each of the following statements is valid, and represents one way in which the corresponding invalid table creation statement could be made to work:

```
CREATE TABLE t1 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  UNIQUE KEY (col1, col2, col3)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  UNIQUE KEY (col1, col3)
)
PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

This example shows the error produced in such cases:

```
mysql> CREATE TABLE t3 (
->   col1 INT NOT NULL,
->   col2 DATE NOT NULL,
->   col3 INT NOT NULL,
->   col4 INT NOT NULL,
->   UNIQUE KEY (col1, col2),
->   UNIQUE KEY (col3)
-> )
-> PARTITION BY HASH(col1 + col3)
-> PARTITIONS 4;
ERROR 1491 (HY000): A PRIMARY KEY MUST INCLUDE ALL COLUMNS IN THE TABLE'S PARTITIONING FUNCTION
```

The **CREATE** statement fails because both **col1** and **col3** are included in the proposed partitioning key, but neither of these columns is part of both of unique keys on the table. This shows one possible fix for the invalid table definition;

```
mysql> CREATE TABLE t3 (
->   col1 INT NOT NULL,
->   col2 DATE NOT NULL,
->   col3 INT NOT NULL,
->   col4 INT NOT NULL,
->   UNIQUE KEY (col1, col2, col3),
->   UNIQUE KEY (col3)
-> )
-> PARTITION BY HASH(col3)
-> PARTITIONS 4;
Query OK, 0 rows affected (0.05 sec)
```

In this case, the proposed partitioning key **col3** is part of both unique keys, and the table creation statement succeeds.

Since every primary key is by definition a unique key, this restriction also includes the table's primary key, if it has one. For example, the next two statements are invalid:

```
CREATE TABLE t4 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  PRIMARY KEY(col1, col2)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t5 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  PRIMARY KEY(col1, col3),
  UNIQUE KEY(col2)
)
PARTITION BY HASH( YEAR(col2) )
PARTITIONS 4;
```

In both cases, the primary key does not include all columns referenced in the partitioning expression. However, both of the next two statements are valid:

```
CREATE TABLE t6 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  PRIMARY KEY(col1, col2)
)
PARTITION BY HASH(col1 + YEAR(col2))
PARTITIONS 4;

CREATE TABLE t7 (
  col1 INT NOT NULL,
  col2 DATE NOT NULL,
  col3 INT NOT NULL,
  col4 INT NOT NULL,
  PRIMARY KEY(col1, col2, col4),
  UNIQUE KEY(col2, col1)
)
PARTITION BY HASH(col1 + YEAR(col2))
PARTITIONS 4;
```

If a table has no unique keys — this includes having no primary key — then this restriction does not apply, and you may use any column or columns in the partitioning expression as long as the column type is compatible with the partitioning type.

For the same reason, you cannot later add a unique key to a partitioned table unless the key includes all columns used by the table's partitioning expression. Consider given the partitioned table defined as shown here:

```
CREATE TABLE t_no_pk (c1 INT, c2 INT)
PARTITION BY RANGE(c1) (
  PARTITION p0 VALUES LESS THAN (10),
  PARTITION p1 VALUES LESS THAN (20),
  PARTITION p2 VALUES LESS THAN (30),
  PARTITION p3 VALUES LESS THAN (40)
);
```

It is possible to add a primary key to `t_no_pk` using either of these `ALTER TABLE` statements:

```
# possible PK
ALTER TABLE t_no_pk ADD PRIMARY KEY(c1);

# also a possible PK
ALTER TABLE t_no_pk ADD PRIMARY KEY(c1, c2);
```

However, the next statement fails, because `c1` is part of the partitioning key, but is not part of the proposed primary key:

```
# fails with ERROR 1482
ALTER TABLE t_no_pk ADD PRIMARY KEY(c2);
```

Since `t_no_pk` has only `c1` in its partitioning expression, attempting to adding a unique key on `c2` alone fails. However, you can add a unique key that uses both `c1` and `c2`.

These rules also apply to existing non-partitioned tables that you wish to partition using `ALTER TABLE ... PARTITION BY`. Consider a table `np_pk` defined as shown here:

```
CREATE TABLE np_pk (
  id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(50),
  added DATE,
  PRIMARY KEY (id)
);
```

The following `ALTER TABLE` statements fails with an error, because the `added` column is not part of any unique key in the table:

```
ALTER TABLE np_pk
PARTITION BY HASH( TO_DAYS(added) )
PARTITIONS 4;
```

This statement, however, would be valid:

```
ALTER TABLE np_pk
PARTITION BY HASH(id)
PARTITIONS 4;
```

In the case of `np_pk`, the only column that may be used as part of a partitioning expression is `id`; if you wish to partition this table using any other column or columns in the partitioning expression, you must first modify the table, either by adding the desired column or columns to the primary key, or by dropping the primary key altogether.

We are working to remove this limitation in a future MySQL release series.

17.5.2. Partitioning Limitations Relating to Storage Engines

The following limitations apply to the use of storage engines with user-defined partitioning of tables.

MERGE storage engine. Tables using the `MERGE` storage engine cannot be partitioned.

FEDERATED storage engine. Partitioning of `FEDERATED` tables is not supported. Beginning with MySQL 5.1.15, it is not possible to create partitioned `FEDERATED` tables at all. We are working to remove this limitation in a future MySQL release.

CSV storage engine. Partitioned tables using the `CSV` storage engine are not supported. Starting with MySQL 5.1.12, it is not possible to create partitioned `CSV` tables at all.

BLACKHOLE storage engine. Prior to MySQL 5.1.6, tables using the `BLACKHOLE` storage engine also could not be partitioned.

NDB storage engine (MySQL Cluster). Partitioning by `KEY` (or `LINEAR KEY`) is the only type of partitioning supported for the `NDB` storage engine. Beginning with MySQL 5.1.12, it is not possible to create a Cluster table using any partitioning type other than `[LINEAR] KEY`, and attempting to do so gives rise to an error.

Upgrading partitioned tables. When performing an upgrade, tables which are partitioned by `KEY` and that use any storage engine other than `NDBCLUSTER` and must be dumped and reloaded.

Same storage engine for all partitions. All of a table's partitions and subpartitions (if there are any of the latter) must use the same storage engine. We are working to remove this limitation in a future MySQL release.

17.5.3. Partitioning Limitations Relating to Functions

This section discusses limitations in MySQL Partitioning relating specifically to functions used in partitioning expressions, including both those function which are specifically supported for use in partitioning expressions, and those which are specifically prohibited.

17.5.3.1. Functions Supported in Partitioning Expressions

Beginning with MySQL 5.1.12, only the following MySQL functions are specifically supported in partitioning expressions:

- `ABS()`
- `CEILING()`

- `DAY()`
- `DAYOFMONTH()`
- `DAYOFWEEK()`
- `DAYOFYEAR()`
- `DATEDIFF()`
- `EXTRACT()`
- `FLOOR()`
- `HOUR()`
- `MICROSECOND()`
- `MINUTE()`
- `MOD()`
- `MONTH()`
- `QUARTER()`
- `SECOND()`
- `TIME_TO_SEC()`
- `TO_DAYS()`
- `WEEKDAY()`
- `YEAR()`
- `YEARWEEK()`

17.5.3.2. Functions Not Permitted in Partitioning Expressions

Beginning with MySQL 5.1.12, the following MySQL functions are specifically not allowed in partitioning expressions:

- `ASCII()`
- `BIT_COUNT()`
- `BIT_LENGTH()`
- `CASE()`
- `CAST()`
- `CHAR_LENGTH()`
- `CHARACTER_LENGTH()`
- `CONVERT()`
- `CRC32()`
- `FIND_IN_SET()`
- `GREATEST()`
- `IFNULL()`

- `INET_ATON()`
- `INSTR()`
- `ISNULL()`
- `LEAST()`
- `LENGTH()`
- `LOCATE()`
- `NULLIF()`
- `OCTET_LENGTH()`
- `ORD()`
- `PERIOD_ADD()`
- `PERIOD_DIFF()`
- `POSITION()`
- `ROUND()`
- `SIGN()`
- `STRCMP()`
- `TIMESTAMPDIFF()`
- `UNIX_TIMESTAMP()`
- `WEEK()`
- `WEEKOFYEAR()`

Chapter 18. Spatial Extensions

MySQL supports spatial extensions to allow the generation, storage, and analysis of geographic features. These features are available for [MyISAM](#), [InnoDB](#), [NDB](#), and [ARCHIVE](#) tables.

For spatial columns, [MyISAM](#) supports both [SPATIAL](#) and non-[SPATIAL](#) indexes. Other storage engines support non-[SPATIAL](#) indexes, as described in [Section 12.1.7, “CREATE INDEX Syntax”](#).

This chapter covers the following topics:

- The basis of these spatial extensions in the OpenGIS geometry model
- Data formats for representing spatial data
- How to use spatial data in MySQL
- Use of indexing for spatial data
- MySQL differences from the OpenGIS specification

Additional resources

- The Open Geospatial Consortium publishes the *OpenGIS® Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at <http://www.opengis.org/docs/99-049.pdf>.
- If you have questions or concerns about the use of the spatial extensions to MySQL, you can discuss them in the GIS forum: <http://forums.mysql.com/list.php?23>.

18.1. Introduction to MySQL Spatial Support

MySQL implements spatial extensions following the specification of the Open Geospatial Consortium (OGC). This is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data. The OGC maintains a Web site at <http://www.opengis.org/>.

In 1997, the Open Geospatial Consortium published the *OpenGIS® Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at <http://www.opengis.org/docs/99-049.pdf>. It contains additional information relevant to this chapter.

MySQL implements a subset of the **SQL with Geometry Types** environment proposed by OGC. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specification describes a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

A **geographic feature** is anything in the world that has a location. A feature can be:

- An entity. For example, a mountain, a pond, a city.
- A space. For example, town district, the tropics.
- A definable location. For example, a crossroad, as a particular place where two streets intersect.

Some documents use the term **geospatial feature** to refer to geographic features.

Geometry is another word that denotes a geographic feature. Originally the word **geometry** meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

This chapter uses all of these terms synonymously: **geographic feature**, **geospatial feature**, **feature**, or **geometry**. Here, the term most

commonly used is **geometry**, defined as *a point or an aggregate of points representing anything in the world that has a location*.

18.2. The OpenGIS Geometry Model

The set of geometry types proposed by OGC's **SQL with Geometry Types** environment is based on the **OpenGIS Geometry Model**. In this model, each geometric object has the following general properties:

- It is associated with a Spatial Reference System, which describes the coordinate space in which the object is defined.
- It belongs to some geometry class.

18.2.1. The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

- **Geometry** (non-instantiable)
 - **Point** (instantiable)
 - **Curve** (non-instantiable)
 - **LineString** (instantiable)
 - **Line**
 - **LinearRing**
 - **Surface** (non-instantiable)
 - **Polygon** (instantiable)
 - **GeometryCollection** (instantiable)
 - **MultiPoint** (instantiable)
 - **MultiCurve** (non-instantiable)
 - **MultiLineString** (instantiable)
 - **MultiSurface** (non-instantiable)
 - **MultiPolygon** (instantiable)

It is not possible to create objects in non-instantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

Geometry is the base class. It is an abstract class. The instantiable subclasses of **Geometry** are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base **Geometry** class has subclasses for **Point**, **Curve**, **Surface**, and **GeometryCollection**:

- **Point** represents zero-dimensional objects.
- **Curve** represents one-dimensional objects, and has subclass **LineString**, with sub-subclasses **Line** and **LinearRing**.
- **Surface** is designed for two-dimensional objects and has subclass **Polygon**.
- **GeometryCollection** has specialized zero-, one-, and two-dimensional collection classes named **MultiPoint**, **MultiLineString**, and **MultiPolygon** for modeling geometries corresponding to collections of **Points**, **LineStrings**, and **Polygons**, respectively. **MultiCurve** and **MultiSurface** are introduced as abstract superclasses that generalize the collection in-

terfaces to handle [Curves](#) and [Surfaces](#).

[Geometry](#), [Curve](#), [Surface](#), [MultiCurve](#), and [MultiSurface](#) are defined as non-instantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

[Point](#), [LineString](#), [Polygon](#), [GeometryCollection](#), [MultiPoint](#), [MultiLineString](#), and [MultiPolygon](#) are instantiable classes.

18.2.2. Class [Geometry](#)

[Geometry](#) is the root class of the hierarchy. It is a non-instantiable class but has a number of properties that are common to all geometry values created from any of the [Geometry](#) subclasses. These properties are described in the following list. Particular subclasses have their own specific properties, described later.

Geometry Properties

A geometry value has the following properties:

- Its **type**. Each geometry belongs to one of the instantiable classes in the hierarchy.
- Its **SRID**, or Spatial Reference Identifier. This value identifies the geometry's associated Spatial Reference System that describes the coordinate space in which the geometry object is defined.

In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

- Its **coordinates** in its Spatial Reference System, represented as double-precision (eight-byte) numbers. All non-empty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the **planar** coordinate system and the distance on the **geocentric** system (coordinates on the Earth's surface) are different things.

- Its **interior**, **boundary**, and **exterior**.

Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.

- Its **MBR** (Minimum Bounding Rectangle), or Envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

```
((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

- Whether the value is **simple** or **non-simple**. Geometry values of types ([LineString](#), [MultiPoint](#), [MultiLineString](#)) are either simple or non-simple. Each type determines its own assertions for being simple or non-simple.
- Whether the value is **closed** or **not closed**. Geometry values of types ([LineString](#), [MultiString](#)) are either closed or not closed. Each type determines its own assertions for being closed or not closed.
- Whether the value is **empty** or **non-empty**. A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a [NULL](#) value). An empty geometry is defined to be always simple and has an area of 0.
- Its **dimension**. A geometry can have a dimension of -1, 0, 1, or 2:
 - -1 for an empty geometry.
 - 0 for a geometry with no length and no area.
 - 1 for a geometry with non-zero length and zero area.
 - 2 for a geometry with non-zero area.

Point objects have a dimension of zero. **LineString** objects have a dimension of 1. **Polygon** objects have a dimension of 2. The dimensions of **MultiPoint**, **MultiLineString**, and **MultiPolygon** objects are the same as the dimensions of the elements they consist of.

18.2.3. Class **Point**

A **Point** is a geometry that represents a single location in coordinate space.

Point Examples

- Imagine a large-scale map of the world with many cities. A **Point** object could represent each city.
- On a city map, a **Point** object could represent a bus stop.

Point Properties

- X-coordinate value.
- Y-coordinate value.
- **Point** is defined as a zero-dimensional geometry.
- The boundary of a **Point** is the empty set.

18.2.4. Class **Curve**

A **Curve** is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of **Curve** define the type of interpolation between points. **Curve** is a non-instantiable class.

Curve Properties

- A **Curve** has the coordinates of its points.
- A **Curve** is defined as a one-dimensional geometry.
- A **Curve** is simple if it does not pass through the same point twice.
- A **Curve** is closed if its start point is equal to its endpoint.
- The boundary of a closed **Curve** is empty.
- The boundary of a non-closed **Curve** consists of its two endpoints.
- A **Curve** that is simple and closed is a **LinearRing**.

18.2.5. Class **LineString**

A **LineString** is a **Curve** with linear interpolation between points.

LineString Examples

- On a world map, **LineString** objects could represent rivers.
- In a city map, **LineString** objects could represent streets.

LineString Properties

- A **LineString** has coordinates of segments, defined by each consecutive pair of points.
- A **LineString** is a **Line** if it consists of exactly two points.
- A **LineString** is a **LinearRing** if it is both closed and simple.

18.2.6. Class Surface

A **Surface** is a two-dimensional geometry. It is a non-instantiable class. Its only instantiable subclass is **Polygon**.

Surface Properties

- A **Surface** is defined as a two-dimensional geometry.
- The OpenGIS specification defines a simple **Surface** as a geometry that consists of a single “patch” that is associated with a single exterior boundary and zero or more interior boundaries.
- The boundary of a simple **Surface** is the set of closed curves corresponding to its exterior and interior boundaries.

18.2.7. Class Polygon

A **Polygon** is a planar **Surface** representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the **Polygon**.

Polygon Examples

- On a region map, **Polygon** objects could represent forests, districts, and so on.

Polygon Assertions

- The boundary of a **Polygon** consists of a set of **LinearRing** objects (that is, **LineString** objects that are both simple and closed) that make up its exterior and interior boundaries.
- A **Polygon** has no rings that cross. The rings in the boundary of a **Polygon** may intersect at a **Point**, but only as a tangent.
- A **Polygon** has no lines, spikes, or punctures.
- A **Polygon** has an interior that is a connected point set.
- A **Polygon** may have holes. The exterior of a **Polygon** with holes is not connected. Each hole defines a connected component of the exterior.

The preceding assertions make a **Polygon** a simple geometry.

18.2.8. Class GeometryCollection

A **GeometryCollection** is a geometry that is a collection of one or more geometries of any class.

All the elements in a **GeometryCollection** must be in the same Spatial Reference System (that is, in the same coordinate system). There are no other constraints on the elements of a **GeometryCollection**, although the subclasses of **GeometryCollection** described in the following sections may restrict membership. Restrictions may be based on:

- Element type (for example, a **MultiPoint** may contain only **Point** elements)

- Dimension
- Constraints on the degree of spatial overlap between elements

18.2.9. Class **MultiPoint**

A **MultiPoint** is a geometry collection composed of **Point** elements. The points are not connected or ordered in any way.

MultiPoint Examples

- On a world map, a **MultiPoint** could represent a chain of small islands.
- On a city map, a **MultiPoint** could represent the outlets for a ticket office.

MultiPoint Properties

- A **MultiPoint** is a zero-dimensional geometry.
- A **MultiPoint** is simple if no two of its **Point** values are equal (have identical coordinate values).
- The boundary of a **MultiPoint** is the empty set.

18.2.10. Class **MultiCurve**

A **MultiCurve** is a geometry collection composed of **Curve** elements. **MultiCurve** is a non-instantiable class.

MultiCurve Properties

- A **MultiCurve** is a one-dimensional geometry.
- A **MultiCurve** is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.
- A **MultiCurve** boundary is obtained by applying the “mod 2 union rule” (also known as the “odd-even rule”): A point is in the boundary of a **MultiCurve** if it is in the boundaries of an odd number of **MultiCurve** elements.
- A **MultiCurve** is closed if all of its elements are closed.
- The boundary of a closed **MultiCurve** is always empty.

18.2.11. Class **MultiLineString**

A **MultiLineString** is a **MultiCurve** geometry collection composed of **LineString** elements.

MultiLineString Examples

- On a region map, a **MultiLineString** could represent a river system or a highway system.

18.2.12. Class **MultiSurface**

A **MultiSurface** is a geometry collection composed of surface elements. **MultiSurface** is a non-instantiable class. Its only instantiable subclass is **MultiPolygon**.

MultiSurface Assertions

- Two [MultiSurface](#) surfaces have no interiors that intersect.
- Two [MultiSurface](#) elements have boundaries that intersect at most at a finite number of points.

18.2.13. Class [MultiPolygon](#)

A [MultiPolygon](#) is a [MultiSurface](#) object composed of [Polygon](#) elements.

[MultiPolygon](#) Examples

- On a region map, a [MultiPolygon](#) could represent a system of lakes.

[MultiPolygon](#) Assertions

- A [MultiPolygon](#) has no two [Polygon](#) elements with interiors that intersect.
- A [MultiPolygon](#) has no two [Polygon](#) elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.
- A [MultiPolygon](#) may not have cut lines, spikes, or punctures. A [MultiPolygon](#) is a regular, closed point set.
- A [MultiPolygon](#) that has more than one [Polygon](#) has an interior that is not connected. The number of connected components of the interior of a [MultiPolygon](#) is equal to the number of [Polygon](#) values in the [MultiPolygon](#).

[MultiPolygon](#) Properties

- A [MultiPolygon](#) is a two-dimensional geometry.
- A [MultiPolygon](#) boundary is a set of closed curves ([LineString](#) values) corresponding to the boundaries of its [Polygon](#) elements.
- Each [Curve](#) in the boundary of the [MultiPolygon](#) is in the boundary of exactly one [Polygon](#) element.
- Every [Curve](#) in the boundary of an [Polygon](#) element is in the boundary of the [MultiPolygon](#).

18.3. Supported Spatial Data Formats

This section describes the standard spatial data formats that are used to represent geometry objects in queries. They are:

- Well-Known Text (WKT) format
- Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format.

18.3.1. Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form.

Examples of WKT representations of geometry objects:

- A [Point](#):

```
POINT(15 20)
```

Note that point coordinates are specified with no separating comma.

- A **LineString** with four points:

```
LINESTRING(0 0, 10 10, 20 25, 50 60)
```

Note that point coordinate pairs are separated by commas.

- A **Polygon** with one exterior ring and one interior ring:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- A **MultiPoint** with three **Point** values:

```
MULTIPOINT(0 0, 20 20, 60 60)
```

- A **MultiLineString** with two **LineString** values:

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- A **MultiPolygon** with two **Polygon** values:

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- A **GeometryCollection** consisting of two **Point** values and one **LineString**:

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

A Backus-Naur grammar that specifies the formal production rules for writing WKT values can be found in the OpenGIS specification document referenced near the beginning of this chapter.

18.3.2. Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation for geometric values is defined by the OpenGIS specification. It is also defined in the ISO *SQL/MM Part 3: Spatial* standard.

WKB is used to exchange geometry data as binary streams represented by **BLOB** values containing geometric WKB information.

WKB uses one-byte unsigned integers, four-byte unsigned integers, and eight-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to **POINT(1 1)** consists of this sequence of 21 bytes (each represented here by two hex digits):

```
010100000000000000000000F03F000000000000F03F
```

The sequence may be broken down into these components:

```
Byte order : 01
WKB type   : 01000000
X          : 000000000000F03F
Y          : 000000000000F03F
```

Component representation is as follows:

- The byte order may be either 0 or 1 to indicate little-endian or big-endian storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.
- The WKB type is a code that indicates the geometry type. Values from 1 through 7 indicate **Point**, **LineString**, **Polygon**, **MultiPoint**, **MultiLineString**, **MultiPolygon**, and **GeometryCollection**.

- A `Point` value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values are represented by more complex data structures, as detailed in the OpenGIS specification.

18.4. Creating a Spatially Enabled MySQL Database

This section describes the data types you can use for representing spatial data in MySQL, and the functions available for creating and retrieving spatial values.

18.4.1. MySQL Spatial Data Types

MySQL has data types that correspond to OpenGIS classes. Some of these types hold single geometry values:

- `GEOMETRY`
- `POINT`
- `LINestring`
- `POLYGON`

`GEOMETRY` can store geometry values of any type. The other single-value types (`POINT`, `LINestring`, and `POLYGON`) restrict their values to a particular geometry type.

The other data types hold collections of values:

- `MULTIPOINT`
- `MULTILINestring`
- `MULTIPOLYGON`
- `GEOMETRYCOLLECTION`

`GEOMETRYCOLLECTION` can store a collection of objects of any type. The other collection types (`MULTIPOINT`, `MULTILINestring`, `MULTIPOLYGON`, and `GEOMETRYCOLLECTION`) restrict collection members to those having a particular geometry type.

18.4.2. Creating Spatial Values

This section describes how to create spatial values using Well-Known Text and Well-Known Binary functions that are defined in the OpenGIS standard, and using MySQL-specific functions.

18.4.2.1. Creating Geometry Values Using WKT Functions

MySQL provides a number of functions that take as input parameters a Well-Known Text representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromText()` accepts a WKT of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- `GeomCollFromText(wkt[,srid]), GeometryCollectionFromText(wkt[,srid])`

Constructs a `GEOMETRYCOLLECTION` value using its WKT representation and SRID.

- `GeomFromText(wkt[,srid]), GeometryFromText(wkt[,srid])`

Constructs a geometry value of any type using its WKT representation and SRID.

- `LineFromText(wkt[,srid]), LineStringFromText(wkt[,srid])`
Constructs a `LINESTRING` value using its WKT representation and SRID.
- `MLineFromText(wkt[,srid]), MultiLineStringFromText(wkt[,srid])`
Constructs a `MULTILINESTRING` value using its WKT representation and SRID.
- `MPointFromText(wkt[,srid]), MultiPointFromText(wkt[,srid])`
Constructs a `MULTIPOINT` value using its WKT representation and SRID.
- `MPolyFromText(wkt[,srid]), MultiPolygonFromText(wkt[,srid])`
Constructs a `MULTIPOLYGON` value using its WKT representation and SRID.
- `PointFromText(wkt[,srid])`
Constructs a `POINT` value using its WKT representation and SRID.
- `PolyFromText(wkt[,srid]), PolygonFromText(wkt[,srid])`
Constructs a `POLYGON` value using its WKT representation and SRID.

The OpenGIS specification also defines the following optional functions, which MySQL does not implement. These functions construct `Polygon` or `MultiPolygon` values based on the WKT representation of a collection of rings or closed `LineString` values. These values may intersect.

- `BdMPolyFromText(wkt,srid)`
Constructs a `MultiPolygon` value from a `MultiLineString` value in WKT format containing an arbitrary collection of closed `LineString` values.
- `BdPolyFromText(wkt,srid)`
Constructs a `Polygon` value from a `MultiLineString` value in WKT format containing an arbitrary collection of closed `LineString` values.

18.4.2.2. Creating Geometry Values Using WKB Functions

MySQL provides a number of functions that take as input parameters a `BLOB` containing a Well-Known Binary representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromWKB()` accepts a WKB of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- `GeomCollFromWKB(wkb[,srid]), GeometryCollectionFromWKB(wkb[,srid])`
Constructs a `GEOMETRYCOLLECTION` value using its WKB representation and SRID.
- `GeomFromWKB(wkb[,srid]), GeometryFromWKB(wkb[,srid])`
Constructs a geometry value of any type using its WKB representation and SRID.
- `LineFromWKB(wkb[,srid]), LineStringFromWKB(wkb[,srid])`
Constructs a `LINESTRING` value using its WKB representation and SRID.
- `MLineFromWKB(wkb[,srid]), MultiLineStringFromWKB(wkb[,srid])`
Constructs a `MULTILINESTRING` value using its WKB representation and SRID.
- `MPointFromWKB(wkb[,srid]), MultiPointFromWKB(wkb[,srid])`

Constructs a **MULTIPOINT** value using its WKB representation and SRID.

- **MPolyFromWKB(wkb[,srid]), MultiPolygonFromWKB(wkb[,srid])**

Constructs a **MULTIPOLYGON** value using its WKB representation and SRID.

- **PointFromWKB(wkb[,srid])**

Constructs a **POINT** value using its WKB representation and SRID.

- **PolyFromWKB(wkb[,srid]), PolygonFromWKB(wkb[,srid])**

Constructs a **POLYGON** value using its WKB representation and SRID.

The OpenGIS specification also describes optional functions for constructing **Polygon** or **MultiPolygon** values based on the WKB representation of a collection of rings or closed **LineString** values. These values may intersect. MySQL does not implement these functions:

- **BdMPolyFromWKB(wkb,srid)**

Constructs a **MultiPolygon** value from a **MultiLineString** value in WKB format containing an arbitrary collection of closed **LineString** values.

- **BdPolyFromWKB(wkb,srid)**

Constructs a **Polygon** value from a **MultiLineString** value in WKB format containing an arbitrary collection of closed **LineString** values.

18.4.2.3. Creating Geometry Values Using MySQL-Specific Functions

MySQL provides a set of useful non-standard functions for creating geometry WKB representations. The functions described in this section are MySQL extensions to the OpenGIS specification. The results of these functions are **BLOB** values containing WKB representations of geometry values with no SRID. The results of these functions can be substituted as the first argument for any function in the **GeomFromWKB()** function family.

- **GeometryCollection(g1,g2,...)**

Constructs a WKB **GeometryCollection**. If any argument is not a well-formed WKB representation of a geometry, the return value is **NULL**.

- **LineString(pt1,pt2,...)**

Constructs a WKB **LineString** value from a number of WKB **Point** arguments. If any argument is not a WKB **Point**, the return value is **NULL**. If the number of **Point** arguments is less than two, the return value is **NULL**.

- **MultiLineString(ls1,ls2,...)**

Constructs a WKB **MultiLineString** value using WKB **LineString** arguments. If any argument is not a WKB **LineString**, the return value is **NULL**.

- **MultiPoint(pt1,pt2,...)**

Constructs a WKB **MultiPoint** value using WKB **Point** arguments. If any argument is not a WKB **Point**, the return value is **NULL**.

- **MultiPolygon(poly1,poly2,...)**

Constructs a WKB **MultiPolygon** value from a set of WKB **Polygon** arguments. If any argument is not a WKB **Polygon**, the return value is **NULL**.

- **Point(x,y)**

Constructs a WKB [Point](#) using its coordinates.

- [Polygon\(*ls1*, *ls2*, ...\)](#)

Constructs a WKB [Polygon](#) value from a number of WKB [LineString](#) arguments. If any argument does not represent the WKB of a [LinearRing](#) (that is, not a closed and simple [LineString](#)) the return value is [NULL](#).

18.4.3. Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with [CREATE TABLE](#) or [ALTER TABLE](#). Currently, spatial columns are supported for [MyISAM](#), [InnoDB](#), [NDB](#), and [ARCHIVE](#) tables. See also the annotations about spatial indexes under [Section 18.6.1, “Creating Spatial Indexes”](#).

- Use the [CREATE TABLE](#) statement to create a table with a spatial column:

```
CREATE TABLE geom (g GEOMETRY);
```

- Use the [ALTER TABLE](#) statement to add or drop a spatial column to or from an existing table:

```
ALTER TABLE geom ADD pt POINT;
ALTER TABLE geom DROP pt;
```

18.4.4. Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values into internal geometry format:

- Perform the conversion directly in the [INSERT](#) statement:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

- Perform the conversion prior to the [INSERT](#):

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

The following examples insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomFromText(@g));
```

The preceding examples all use [GeomFromText\(\)](#) to create geometry values. You can also use type-specific functions:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
```

```
INSERT INTO geom VALUES (PolygonFromText(@g));  
SET @g =  
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';  
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

Note that if a client application program wants to use WKB representations of geometry values, it is responsible for sending correctly formed WKB in queries to the server. However, there are several ways of satisfying this requirement. For example:

- Inserting a `POINT(1 1)` value with hex literal syntax:

```
mysql> INSERT INTO geom VALUES  
-> (GeomFromWKB(0x010100000000000000000000F03F0000000000F03F));
```

- An ODBC application can send a WKB representation, binding it to a placeholder using an argument of `BLOB` type:

```
INSERT INTO geom VALUES (GeomFromWKB(?))
```

Other programming interfaces may support a similar placeholder mechanism.

- In a C program, you can escape a binary value using `mysql_real_escape_string()` and include the result in a query string that is sent to the server. See [Section 25.2.3.53](#), “`mysql_real_escape_string()`”.

18.4.5. Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them into WKT or WKB format.

- Fetching spatial data in internal format:

Fetching geometry values using internal format can be useful in table-to-table transfers:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

- Fetching spatial data in WKT format:

The `AsText()` function converts a geometry from internal format into a WKT string.

```
SELECT AsText(g) FROM geom;
```

- Fetching spatial data in WKB format:

The `AsBinary()` function converts a geometry from internal format into a `BLOB` containing the WKB value.

```
SELECT AsBinary(g) FROM geom;
```

18.5. Analyzing Spatial Information

After populating spatial columns with values, you are ready to query and analyze them. MySQL provides a set of functions to perform various operations on spatial data. These functions can be grouped into four major categories according to the type of operation they perform:

- Functions that convert geometries between various formats
- Functions that provide access to qualitative or quantitative properties of a geometry
- Functions that describe relations between two geometries
- Functions that create new geometries from existing ones

Spatial analysis functions can be used in many contexts, such as:

- Any interactive SQL program, such as [mysql](#) or MySQL Query Browser
- Application programs written in any language that supports a MySQL client API

18.5.1. Geometry Format Conversion Functions

MySQL supports the following functions for converting geometry values between internal format and either WKT or WKB format:

- [AsBinary\(*g*\)](#)

Converts a value in internal geometry format to its WKB representation and returns the binary result.

```
SELECT AsBinary(g) FROM geom;
```

- [AsText\(*g*\)](#)

Converts a value in internal geometry format to its WKT representation and returns the string result.

```
mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(GeomFromText(@g));
+-----+
| AsText(GeomFromText(@g)) |
+-----+
| LINESTRING(1 1,2 2,3 3) |
+-----+
```

- [GeomFromText\(*wkt*\[,*srid*\]\)](#)

Converts a string value from its WKT representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as [PointFromText\(\)](#) and [LineFromText\(\)](#). See [Section 18.4.2.1, “Creating Geometry Values Using WKT Functions”](#).

- [GeomFromWKB\(*wkb*\[,*srid*\]\)](#)

Converts a binary value from its WKB representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as [PointFromWKB\(\)](#) and [LineFromWKB\(\)](#). See [Section 18.4.2.2, “Creating Geometry Values Using WKB Functions”](#).

18.5.2. Geometry Functions

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such functions return [NULL](#) if the argument is of an incorrect geometry type. For example, [Area\(\)](#) returns [NULL](#) if the object type is neither [Polygon](#) nor [MultiPolygon](#).

18.5.2.1. General Geometry Functions

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

- [Dimension\(*g*\)](#)

Returns the inherent dimension of the geometry value *g*. The result can be -1, 0, 1, or 2. The meaning of these values is given in [Section 18.2.2, “Class Geometry”](#).

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| 1 |
+-----+
```

- **Envelope(*g*)**

Returns the Minimum Bounding Rectangle (MBR) for the geometry value *g*. The result is returned as a **Polygon** value.

The polygon is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

```
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)')));
+-----+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| POLYGON((1 1,2 1,2 2,1 2,1 1)) |
+-----+
```

- **GeometryType(*g*)**

Returns as a string the name of the geometry type of which the geometry instance *g* is a member. The name corresponds to one of the instantiable **Geometry** subclasses.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+-----+
| GeometryType(GeomFromText('POINT(1 1)')) |
+-----+
| POINT |
+-----+
```

- **SRID(*g*)**

Returns an integer indicating the Spatial Reference System ID for the geometry value *g*.

In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+-----+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----+
| 101 |
+-----+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- **Boundary(*g*)**

Returns a geometry that is the closure of the combinatorial boundary of the geometry value *g*.

- **IsEmpty(*g*)**

Returns 1 if the geometry value *g* is the empty geometry, 0 if it is not empty, and -1 if the argument is **NULL**. If the geometry is empty, it represents the empty point set.

- **IsSimple(*g*)**

Currently, this function is a placeholder and should not be used. If implemented, its behavior will be as described in the next paragraph.

Returns 1 if the geometry value *g* has no anomalous geometric points, such as self-intersection or self-tangency. **IsSimple()** returns 0 if the argument is not simple, and -1 if it is **NULL**.

The description of each instantiable geometric class given earlier in the chapter includes the specific conditions that cause an instance of that class to be classified as not simple. (See [Section 18.2.1, “The Geometry Class Hierarchy”](#).)

18.5.2.2. Point Functions

A **Point** consists of X and Y coordinates, which may be obtained using the following functions:

- **X(*p*)**

Returns the X-coordinate value for the point *p* as a double-precision number.

```
mysql> SET @pt = 'Point(56.7 53.34)';
mysql> SELECT X(GeomFromText(@pt));
+-----+
| X(GeomFromText(@pt)) |
+-----+
|          56.7         |
+-----+
```

- **Y(*p*)**

Returns the Y-coordinate value for the point *p* as a double-precision number.

```
mysql> SET @pt = 'Point(56.7 53.34)';
mysql> SELECT Y(GeomFromText(@pt));
+-----+
| Y(GeomFromText(@pt)) |
+-----+
|          53.34        |
+-----+
```

18.5.2.3. LineString Functions

A **LineString** consists of **Point** values. You can extract particular points of a **LineString**, count the number of points that it contains, or obtain its length.

- **EndPoint(*ls*)**

Returns the **Point** that is the endpoint of the **LineString** value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(EndPoint(GeomFromText(@ls)));
+-----+
| AsText(EndPoint(GeomFromText(@ls))) |
+-----+
| POINT(3 3)                          |
+-----+
```

- **GLength(*ls*)**

Returns as a double-precision number the length of the **LineString** value *ls* in its associated spatial reference.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+-----+
| GLength(GeomFromText(@ls)) |
+-----+
|          2.8284271247462     |
+-----+
```

GLength() is a non-standard name. It corresponds to the OpenGIS **Length()** function.

- **NumPoints(*ls*)**

Returns the number of **Point** objects in the **LineString** value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT NumPoints(GeomFromText(@ls));
+-----+
| NumPoints(GeomFromText(@ls)) |
+-----+
|          3                    |
+-----+
```


- `PointN(ls,N)`

Returns the *N*-th `Point` in the `LineString` value *ls*. Points are numbered beginning with 1.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(PointN(GeomFromText(@ls),2));
+-----+
| AsText(PointN(GeomFromText(@ls),2)) |
+-----+
| POINT(2 2) |
+-----+
```

- `StartPoint(ls)`

Returns the `Point` that is the start point of the `LineString` value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(StartPoint(GeomFromText(@ls)));
+-----+
| AsText(StartPoint(GeomFromText(@ls))) |
+-----+
| POINT(1 1) |
+-----+
```

The OpenGIS specification also defines the following function, which MySQL does not implement:

- `IsRing(ls)`

Returns 1 if the `LineString` value *ls* is closed (that is, its `StartPoint()` and `EndPoint()` values are the same) and is simple (does not pass through the same point more than once). Returns 0 if *ls* is not a ring, and -1 if it is `NULL`.

18.5.2.4. MultiLineString Functions

- `GLength(mls)`

Returns as a double-precision number the length of the `MultiLineString` value *mls*. The length of *mls* is equal to the sum of the lengths of its elements.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT GLength(GeomFromText(@mls));
+-----+
| GLength(GeomFromText(@mls)) |
+-----+
| 4.2426406871193 |
+-----+
```

`GLength()` is a non-standard name. It corresponds to the OpenGIS `Length()` function.

- `IsClosed(mls)`

Returns 1 if the `MultiLineString` value *mls* is closed (that is, the `StartPoint()` and `EndPoint()` values are the same for each `LineString` in *mls*). Returns 0 if *mls* is not closed, and -1 if it is `NULL`.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT IsClosed(GeomFromText(@mls));
+-----+
| IsClosed(GeomFromText(@mls)) |
+-----+
| 0 |
+-----+
```

18.5.2.5. Polygon Functions

- `Area(poly)`

Returns as a double-precision number the area of the **Polygon** value *poly*, as measured in its spatial reference system.

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1));'
mysql> SELECT Area(GeomFromText(@poly));
+-----+
| Area(GeomFromText(@poly)) |
+-----+
| 4 |
+-----+
```

- **ExteriorRing(*poly*)**

Returns the exterior ring of the **Polygon** value *poly* as a **LineString**.

```
mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1));'
mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-----+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-----+
| LINESTRING(0 0,0 3,3 3,3 0,0 0) |
+-----+
```

- **InteriorRingN(*poly*,*N*)**

Returns the *N*-th interior ring for the **Polygon** value *poly* as a **LineString**. Rings are numbered beginning with 1.

```
mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1));'
mysql> SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+-----+
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+-----+
| LINESTRING(1 1,1 2,2 2,2 1,1 1) |
+-----+
```

- **NumInteriorRings(*poly*)**

Returns the number of interior rings in the **Polygon** value *poly*.

```
mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1));'
mysql> SELECT NumInteriorRings(GeomFromText(@poly));
+-----+
| NumInteriorRings(GeomFromText(@poly)) |
+-----+
| 1 |
+-----+
```

18.5.2.6. MultiPolygon Functions

- **Area(*mpoly*)**

Returns as a double-precision number the area of the **MultiPolygon** value *mpoly*, as measured in its spatial reference system.

```
mysql> SET @mpoly =
-> 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT Area(GeomFromText(@mpoly));
+-----+
| Area(GeomFromText(@mpoly)) |
+-----+
| 8 |
+-----+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- **Centroid(*mpoly*)**

Returns the mathematical centroid for the **MultiPolygon** value *mpoly* as a **Point**. The result is not guaranteed to be on the **MultiPolygon**.

- **PointOnSurface(*mpoly*)**

Returns a **Point** value that is guaranteed to be on the **MultiPolygon** value *mpoly*.

18.5.2.7. GeometryCollection Functions

- **GeometryN(*gc*, *N*)**

Returns the *N*-th geometry in the **GeometryCollection** value *gc*. Geometries are numbered beginning with 1.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));
+-----+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+-----+
| POINT(1 1)                             |
+-----+
```

- **NumGeometries(*gc*)**

Returns the number of geometries in the **GeometryCollection** value *gc*.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT NumGeometries(GeomFromText(@gc));
+-----+
| NumGeometries(GeomFromText(@gc)) |
+-----+
| 2 |
+-----+
```

18.5.3. Functions That Create New Geometries from Existing Ones

18.5.3.1. Geometry Functions That Produce New Geometries

Section 18.5.2, “Geometry Functions”, discusses several functions that construct new geometries from existing ones. See that section for descriptions of these functions:

- **Envelope(*g*)**
- **StartPoint(*ls*)**
- **EndPoint(*ls*)**
- **PointN(*ls*, *N*)**
- **ExteriorRing(*poly*)**
- **InteriorRingN(*poly*, *N*)**
- **GeometryN(*gc*, *N*)**

18.5.3.2. Spatial Operators

OpenGIS proposes a number of other functions that can produce geometries. They are designed to implement spatial operators.

These functions are not implemented in MySQL. They may appear in future releases.

- **Buffer(*g*, *d*)**
Returns a geometry that represents all points whose distance from the geometry value *g* is less than or equal to a distance of *d*.
- **ConvexHull(*g*)**
Returns a geometry that represents the convex hull of the geometry value *g*.
- **Difference(*g1*, *g2*)**
Returns a geometry that represents the point set difference of the geometry value *g1* with *g2*.
- **Intersection(*g1*, *g2*)**
Returns a geometry that represents the point set intersection of the geometry values *g1* with *g2*.
- **SymDifference(*g1*, *g2*)**
Returns a geometry that represents the point set symmetric difference of the geometry value *g1* with *g2*.
- **Union(*g1*, *g2*)**
Returns a geometry that represents the point set union of the geometry values *g1* and *g2*.

18.5.4. Functions for Testing Spatial Relations Between Geometric Objects

The functions described in these sections take two geometries as input parameters and return a qualitative or quantitative relation between them.

18.5.5. Relations on Geometry Minimal Bounding Rectangles (MBRs)

MySQL provides several functions that test relations between minimal bounding rectangles of two geometries *g1* and *g2*. The return values 1 and 0 indicate true and false, respectively.

- **MBRContains(*g1*, *g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* contains the Minimum Bounding Rectangle of *g2*. This tests the opposite relationship as **MBRWithin()**.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,0 0))');
mysql> SET @g2 = GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
+-----+-----+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+-----+-----+
| 1 | 0 |
+-----+-----+
```

- **MBRDisjoint(*g1*, *g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

- **MBREqual(*g1*, *g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are the same.

- **MBRIntersects(*g1*, *g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* intersect.

- **MBROverlaps(*g1*, *g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* overlap. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either

of the given geometries.

- **MBRTouches(*g1*,*g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* touch. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- **MBRWithin(*g1*,*g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* is within the Minimum Bounding Rectangle of *g2*. This tests the opposite relationship as **MBRContains()**.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+-----+-----+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+-----+-----+
| 1 | 0 |
+-----+-----+
```

18.5.6. Functions That Test Spatial Relationships Between Geometries

The OpenGIS specification defines the following functions. They test the relationship between two geometry values *g1* and *g2*.

The return values 1 and 0 indicate true and false, respectively.

Note

Currently, MySQL does not implement these functions according to the specification. Those that are implemented return the same result as the corresponding MBR-based functions. This includes functions in the following list other than **Distance()** and **Related()**.

These functions may be implemented in future releases with full support for spatial analysis, not just MBR-based support.

- **Contains(*g1*,*g2*)**

Returns 1 or 0 to indicate whether *g1* completely contains *g2*. This tests the opposite relationship as **Within()**.

- **Crosses(*g1*,*g2*)**

Returns 1 if *g1* spatially crosses *g2*. Returns **NULL** if *g1* is a **Polygon** or a **MultiPolygon**, or if *g2* is a **Point** or a **MultiPoint**. Otherwise, returns 0.

The term *spatially crosses* denotes a spatial relation between two given geometries that has the following properties:

- The two geometries intersect
- Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
- Their intersection is not equal to either of the two given geometries

- **Disjoint(*g1*,*g2*)**

Returns 1 or 0 to indicate whether *g1* is spatially disjoint from (does not intersect) *g2*.

- **Distance(*g1*,*g2*)**

Returns as a double-precision number the shortest distance between any two points in the two geometries.

- **Equals(*g1*,*g2*)**

Returns 1 or 0 to indicate whether *g1* is spatially equal to *g2*.

- `Intersects(g1,g2)`

Returns 1 or 0 to indicate whether *g1* spatially intersects *g2*.

- `Overlaps(g1,g2)`

Returns 1 or 0 to indicate whether *g1* spatially overlaps *g2*. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `Related(g1,g2,pattern_matrix)`

Returns 1 or 0 to indicate whether the spatial relationship specified by *pattern_matrix* exists between *g1* and *g2*. Returns -1 if the arguments are `NULL`. The pattern matrix is a string. Its specification will be noted here if this function is implemented.

- `Touches(g1,g2)`

Returns 1 or 0 to indicate whether *g1* spatially touches *g2*. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `Within(g1,g2)`

Returns 1 or 0 to indicate whether *g1* is spatially within *g2*. This tests the opposite relationship as `Contains()`.

18.6. Optimizing Spatial Analysis

Search operations in non-spatial databases can be optimized using `SPATIAL` indexes. This is true for spatial databases as well. With the help of a great variety of multi-dimensional indexing methods that have previously been designed, it is possible to optimize spatial searches. The most typical of these are:

- Point queries that search for all objects that contain a given point
- Region queries that search for all objects that overlap a given region

MySQL uses **R-Trees with quadratic splitting** for `SPATIAL` indexes on spatial columns. A `SPATIAL` index is built using the MBR of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical line-string, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

It is also possible to create normal indexes on spatial columns. In a non-`SPATIAL` index, you must declare a prefix for any spatial column except for `POINT` columns.

`MyISAM` supports both `SPATIAL` and non-`SPATIAL` indexes. Other storage engines support non-`SPATIAL` indexes, as described in [Section 12.1.7, “CREATE INDEX Syntax”](#).

18.6.1. Creating Spatial Indexes

MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but extended with the `SPATIAL` keyword. Currently, columns in spatial indexes must be declared `NOT NULL`. The following examples demonstrate how to create spatial indexes:

- With `CREATE TABLE`:

```
CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
```

- With `ALTER TABLE`:

```
ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- With `CREATE INDEX`:

```
CREATE SPATIAL INDEX sp_index ON geom (g);
```

For **MyISAM** tables, **SPATIAL INDEX** creates an R-tree index. For storage engines that support non-spatial indexing of spatial columns, the engine creates a B-tree index. A B-tree index on spatial values will be useful for exact-value lookups, but not for range scans.

For more information on indexing spatial columns, see [Section 12.1.7, “CREATE INDEX Syntax”](#).

To drop spatial indexes, use **ALTER TABLE** or **DROP INDEX**:

- With **ALTER TABLE**:

```
ALTER TABLE geom DROP INDEX g;
```

- With **DROP INDEX**:

```
DROP INDEX sp_index ON geom;
```

Example: Suppose that a table **geom** contains more than 32,000 geometries, which are stored in the column **g** of type **GEOMETRY**. The table also has an **AUTO_INCREMENT** column **fid** for storing object ID values.

```
mysql> DESCRIBE geom;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| fid   | int(11) |      | PRI | NULL    | auto_increment |
| g     | geometry |      |     |         |                |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+-----+
| count(*) |
+-----+
|      32376 |
+-----+
1 row in set (0.00 sec)
```

To add a spatial index on the column **g**, use this statement:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376 Duplicates: 0 Warnings: 0
```

18.6.2. Using a Spatial Index

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as **MBRContains()** or **MBRWithin()** in the **WHERE** clause. The following query finds all objects that are in the given rectangle:

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
              31000 15000,
              31000 16000,
              30000 16000,
              30000 15000));'
mysql> SELECT fid, AsText(g) FROM geom WHERE
-> MBRContains(GeomFromText(@poly), g);
+-----+-----+
| fid | AsText(g) |
+-----+-----+
| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136. ... |
| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
```

```

+----+-----+
| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
+----+-----+
20 rows in set (0.00 sec)

```

Use [EXPLAIN](#) to check the way this query is executed:

```

mysql> SET @poly =
-> 'Polygon((30000 15000,
           31000 15000,
           31000 16000,
           30000 16000,
           30000 15000));'
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
-> MBRContains(GeomFromText(@poly),g)\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: geom
        type: range
possible_keys: g
         key: g
        key_len: 32
         ref: NULL
        rows: 50
   Extra: Using where
1 row in set (0.00 sec)

```

Check what would happen without a spatial index:

```

mysql> SET @poly =
-> 'Polygon((30000 15000,
           31000 15000,
           31000 16000,
           30000 16000,
           30000 15000));'
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
-> MBRContains(GeomFromText(@poly),g)\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: geom
        type: ALL
possible_keys: NULL
         key: NULL
        key_len: NULL
         ref: NULL
        rows: 32376
   Extra: Using where
1 row in set (0.00 sec)

```

Executing the [SELECT](#) statement without the spatial index yields the same result but causes the execution time to rise from 0.00 seconds to 0.46 seconds:

```

mysql> SET @poly =
-> 'Polygon((30000 15000,
           31000 15000,
           31000 16000,
           30000 16000,
           30000 15000));'
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
-> MBRContains(GeomFromText(@poly),g);
+----+-----+
| fid | AsText(g) |
+----+-----+
| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
+----+-----+

```



```
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
+-----+
20 rows in set (0.46 sec)
```

In future releases, spatial indexes may also be used for optimizing other functions. See [Section 18.5.4, “Functions for Testing Spatial Relations Between Geometric Objects”](#).

18.7. MySQL Conformance and Compatibility

MySQL does not yet implement the following GIS features:

- Additional Metadata Views

OpenGIS specifications propose several additional metadata views. For example, a system view named `GEOMETRY_COLUMNS` contains a description of geometry columns, one row for each geometry column in the database.

- The OpenGIS function `Length()` on `LineString` and `MultiLineString` currently should be called in MySQL as `GLength()`

The problem is that there is an existing SQL function `Length()` that calculates the length of string values, and sometimes it is not possible to distinguish whether the function is called in a textual or spatial context. We need either to solve this somehow, or decide on another function name.

Chapter 19. Stored Procedures and Functions

Stored routines (procedures and functions) are supported in MySQL 5.1. A stored procedure is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored procedure instead.

Answers to some questions that are commonly asked regarding stored routines in MySQL can be found in [Section A.4, “MySQL 5.1 FAQ — Stored Procedures”](#).

MySQL Enterprise

For expert advice on using stored procedures and functions subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

Some situations where stored routines can be particularly useful:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
- When security is paramount. Banks, for example, use stored procedures and functions for all common operations. This provides a consistent and secure environment, and routines can ensure that each operation is properly logged. In such a setup, applications and users would have no access to the database tables directly, but can only execute specific stored routines.

Stored routines can provide improved performance because less information needs to be sent between the server and the client. The tradeoff is that this does increase the load on the database server because more of the work is done on the server side and less is done on the client (application) side. Consider this if many client machines (such as Web servers) are serviced by only one or a few database servers.

Stored routines also allow you to have libraries of functions in the database server. This is a feature shared by modern application languages that allow such design internally (for example, by using classes). Using these client application language features is beneficial for the programmer even outside the scope of database use.

MySQL follows the SQL:2003 syntax for stored routines, which is also used by IBM's DB2.

The MySQL implementation of stored routines is still in progress. All syntax described in this chapter is supported and any limitations and extensions are documented where appropriate. Further discussion of restrictions on use of stored routines is given in [Section D.1, “Restrictions on Stored Routines and Triggers”](#).

Binary logging for stored routines takes place as described in [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

Recursive stored procedures are disabled by default, but can be enabled on the server by setting the `max_sp_recursion_depth` server system variable to a nonzero value. See [Section 5.2.3, “System Variables”](#), for more information.

Stored functions cannot be recursive. See [Section D.1, “Restrictions on Stored Routines and Triggers”](#).

19.1. Stored Routines and the Grant Tables

Stored routines require the `proc` table in the `mysql` database. This table is created during the MySQL 5.1 installation procedure. If you are upgrading to MySQL 5.1 from an earlier version, be sure to update your grant tables to make sure that the `proc` table exists. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

The server manipulates the `mysql.proc` table in response to statements that create, alter, or drop stored routines. It is not supported that the server will notice manual manipulation of this table.

The MySQL grant system takes stored routines into account as follows:

- The `CREATE ROUTINE` privilege is needed to create stored routines.
- The `ALTER ROUTINE` privilege is needed to alter or drop stored routines. This privilege is granted automatically to the creator of a routine if necessary, and dropped when the routine creator drops the routine.
- The `EXECUTE` privilege is required to execute stored routines. However, this privilege is granted automatically to the creator of a

routine if necessary (and dropped when the creator drops the routine). Also, the default `SQL SECURITY` characteristic for a routine is `DEFINER`, which enables users who have access to the database with which the routine is associated to execute the routine.

- If the `automatic_sp_privileges` system variable is 0, the `EXECUTE` and `ALTER ROUTINE` privileges are not automatically granted and dropped.

19.2. Stored Routine Syntax

A stored routine is either a procedure or a function. Stored routines are created with `CREATE PROCEDURE` and `CREATE FUNCTION` statements. A procedure is invoked using a `CALL` statement, and can only pass back values using output variables. A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar value. Stored routines may call other stored routines.

A stored procedure or function is associated with a particular database. This has several implications:

- When the routine is invoked, an implicit `USE db_name` is performed (and undone when the routine terminates). `USE` statements within stored routines are disallowed.
- You can qualify routine names with the database name. This can be used to refer to a routine that is not in the current database. For example, to invoke a stored procedure `p` or function `f` that is associated with the `test` database, you can say `CALL test.p()` or `test.f()`.
- When a database is dropped, all stored routines associated with it are dropped as well.

MySQL supports the very useful extension that allows the use of regular `SELECT` statements (that is, without using cursors or local variables) inside a stored procedure. The result set of such a query is simply sent directly to the client. Multiple `SELECT` statements generate multiple result sets, so the client must use a MySQL client library that supports multiple result sets. This means the client must use a client library from a version of MySQL at least as recent as 4.1. The client should also specify the `CLIENT_MULTI_RESULTS` option when it connects. For C programs, this can be done with the `mysql_real_connect()` C API function. See [Section 25.2.3.52](#), “`mysql_real_connect()`”, and [Section 25.2.9](#), “C API Handling of Multiple Statement Execution”.

MySQL Enterprise

MySQL Enterprise subscribers will find numerous articles about stored routines in the MySQL Enterprise Knowledge Base. Access to this collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

The following sections describe the syntax used to create, alter, drop, and invoke stored procedures and functions.

19.2.1. CREATE PROCEDURE and CREATE FUNCTION Syntax

```
CREATE
[DEFINER = { user | CURRENT_USER }]
PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...] routine_body

CREATE
[DEFINER = { user | CURRENT_USER }]
FUNCTION sp_name ([func_parameter[,...]])
RETURNS type
[characteristic ...] routine_body

proc_parameter:
[ IN | OUT | INOUT ] param_name type

func_parameter:
param_name type

type:
Any valid MySQL data type

characteristic:
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'

routine_body:
Valid SQL procedure statement
```

These statements create stored routines. To use them, it is necessary to have the `CREATE ROUTINE` privilege. If binary logging is enabled, the `CREATE FUNCTION` statement might also require the `SUPER` privilege, as described in [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#). MySQL automatically grants the `ALTER ROUTINE` and `EXECUTE` privileges to the routine creator.

By default, the routine is associated with the default database. To associate the routine explicitly with a given database, specify the name as `db_name.sp_name` when you create it.

If the routine name is the same as the name of a built-in SQL function, you must use a space between the name and the following parenthesis when defining the routine, or a syntax error occurs. This is also true when you invoke the routine later. For this reason, we suggest that it is better to avoid re-using the names of existing SQL functions for your own stored routines.

The `IGNORE_SPACE` SQL mode applies to built-in functions, not to stored routines. It is always allowable to have spaces after a routine name, regardless of whether `IGNORE_SPACE` is enabled.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of `()` should be used.

Each parameter can be declared to use any valid data type, except that the `COLLATE` attribute cannot be used.

Each parameter is an `IN` parameter by default. To specify otherwise for a parameter, use the keyword `OUT` or `INOUT` before the parameter name.

Note

Specifying a parameter as `IN`, `OUT`, or `INOUT` is valid only for a `PROCEDURE`. (`FUNCTION` parameters are always regarded as `IN` parameters.)

An `IN` parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns. An `OUT` parameter passes a value from the procedure back to the caller. Its initial value is `NULL` within the procedure, and its value is visible to the caller when the procedure returns. An `INOUT` parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

For each `OUT` or `INOUT` parameter, pass a user-defined variable so that you can obtain its value when the procedure returns. (For an example, see [Section 19.2.4, “CALL Statement Syntax”](#).) If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.

The `RETURNS` clause may be specified only for a `FUNCTION`, for which it is mandatory. It indicates the return type of the function, and the function body must contain a `RETURN value` statement. If the `RETURN` statement returns a value of a different type, the value is coerced to the proper type. For example, if a function specifies an `ENUM` or `SET` value in the `RETURNS` clause, but the `RETURN` statement returns an integer, the value returned from the function is the string for the corresponding `ENUM` member of set of `SET` members.

The `routine_body` consists of a valid SQL procedure statement. This can be a simple statement such as `SELECT` or `INSERT`, or it can be a compound statement written using `BEGIN` and `END`. Compound statement syntax is described in [Section 19.2.5, “BEGIN . . . END Compound Statement Syntax”](#). Compound statements can contain declarations, loops, and other control structure statements. The syntax for these statements is described later in this chapter. See, for example, [Section 19.2.6, “DECLARE Statement Syntax”](#), and [Section 19.2.10, “Flow Control Constructs”](#). Some statements are not allowed in stored routines; see [Section D.1, “Restrictions on Stored Routines and Triggers”](#).

MySQL stores the `sql_mode` system variable setting that is in effect at the time a routine is created, and always executes the routine with this setting in force, *regardless of the current server SQL mode*.

The `CREATE FUNCTION` statement was used in earlier versions of MySQL to support UDFs (user-defined functions). See [Section 28.3, “Adding New Functions to MySQL”](#). UDFs continue to be supported, even with the existence of stored functions. A UDF can be regarded as an external stored function. However, do note that stored functions share their namespace with UDFs. See [Section 8.2.4, “Function Name Parsing and Resolution”](#), for the rules describing how the server interprets references to different kinds of functions.

A procedure or function is considered “deterministic” if it always produces the same result for the same input parameters, and “not deterministic” otherwise. If neither `DETERMINISTIC` nor `NOT DETERMINISTIC` is given in the routine definition, the default is `NOT DETERMINISTIC`.

A routine that contains the `NOW()` function (or its synonyms) or `RAND()` is non-deterministic, but it might still be replication-safe. For `NOW()`, the binary log includes the timestamp and replicates correctly. `RAND()` also replicates correctly as long as it is invoked only once within a routine. (You can consider the routine execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

Currently, the **DETERMINISTIC** characteristic is accepted, but not yet used by the optimizer. However, if binary logging is enabled, this characteristic affects which routine definitions MySQL accepts. See [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

Several characteristics provide information about the nature of data use by the routine. In MySQL, these characteristics are advisory only. The server does not use them to constrain what kinds of statements a routine will be allowed to execute.

- **CONTAINS SQL** indicates that the routine does not contain statements that read or write data. This is the default if none of these characteristics is given explicitly. Examples of such statements are **SET @x = 1** or **DO RELEASE_LOCK('abc')**, which execute but neither read nor write data.
- **NO SQL** indicates that the routine contains no SQL statements.
- **READS SQL DATA** indicates that the routine contains statements that read data (for example, **SELECT**), but not statements that write data.
- **MODIFIES SQL DATA** indicates that the routine contains statements that may write data (for example, **INSERT** or **DELETE**).

The **SQL SECURITY** characteristic can be used to specify whether the routine should be executed using the permissions of the user who creates the routine or the user who invokes it. The default value is **DEFINER**. This feature is new in SQL:2003. The creator or invoker must have permission to access the database with which the routine is associated. It is necessary to have the **EXECUTE** privilege to be able to execute the routine. The user that must have this privilege is either the definer or invoker, depending on how the **SQL SECURITY** characteristic is set.

The optional **DEFINER** clause specifies the MySQL account to be used when checking access privileges at routine execution time for routines that have the **SQL SECURITY DEFINER** characteristic. The **DEFINER** clause was added in MySQL 5.1.8.

If a *user* value is given, it should be a MySQL account in '*user_name*'@'*host_name*' format (the same format used in the **GRANT** statement). The *user_name* and *host_name* values both are required. **CURRENT_USER** also can be given as **CURRENT_USER()**. The default **DEFINER** value is the user who executes the **CREATE PROCEDURE** or **CREATE FUNCTION** or statement. (This is the same as **DEFINER = CURRENT_USER**.)

If you specify the **DEFINER** clause, you cannot set the value to any account but your own unless you have the **SUPER** privilege. These rules determine the legal **DEFINER** user values:

- If you do not have the **SUPER** privilege, the only legal *user* value is your own account, either specified literally or by using **CURRENT_USER**. You cannot set the definer to some other account.
- If you have the **SUPER** privilege, you can specify any syntactically legal account name. If the account does not actually exist, a warning is generated.

Although it is possible to create routines with a non-existent **DEFINER** value, an error occurs if the routine executes with definer privileges but the definer does not exist at execution time.

When the routine is invoked, an implicit **USE db_name** is performed (and undone when the routine terminates). **USE** statements within stored routines are disallowed.

The server uses the data type of a routine parameter or function return value as follows. These rules also apply to local routine variables created with the **DECLARE** statement ([Section 19.2.7.1, “DECLARE Local Variables”](#)).

- Assignments are checked for data type mismatches and overflow. Conversion and overflow problems result in warnings, or errors in strict mode.
- For character data types, if there is a **CHARACTER SET** clause in the declaration, the specified character set and its default collation are used. If there is no such clause, the database character set and collation that are in effect at the time the routine is created are used. (These are given by the values of the **character_set_database** and **collation_database** system variables.) The **COLLATE** attribute is not supported. (This includes use of **BINARY**, because in this context **BINARY** specifies the binary collation of the character set.)
- Only scalar values can be assigned to parameters or variables. For example, a statement such as **SET x = (SELECT 1, 2)** is invalid.

The **COMMENT** clause is a MySQL extension, and may be used to describe the stored routine. This information is displayed by the **SHOW CREATE PROCEDURE** and **SHOW CREATE FUNCTION** statements.

MySQL allows routines to contain DDL statements, such as **CREATE** and **DROP**. MySQL also allows stored procedures (but not stored functions) to contain SQL transaction statements such as **COMMIT**. Stored functions may not contain statements that do explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to allow them.

Stored routines cannot use **LOAD DATA INFILE**.

Statements that return a result set cannot be used within a stored function. This includes **SELECT** statements that do not use **INTO** to fetch column values into variables, **SHOW** statements, and other statements such as **EXPLAIN**. For statements that can be determined at function definition time to return a result set, a **Not allowed to return a result set from a function** error occurs (**ER_SP_NO_RETSET**). For statements that can be determined only at runtime to return a result set, a **PROCEDURE %s can't return a result set in the given context** error occurs (**ER_SP_BADSELECT**).

The following is an example of a simple stored procedure that uses an **OUT** parameter. The example uses the **mysql** client **delimiter** command to change the statement delimiter from **;** to **//** while the procedure is being defined. This allows the **;** delimiter used in the procedure body to be passed through to the server rather than being interpreted by **mysql** itself.

```
mysql> delimiter //
mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
-> BEGIN
->   SELECT COUNT(*) INTO param1 FROM t;
-> END;
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+-----+
| @a    |
+-----+
| 3     |
+-----+
1 row in set (0.00 sec)
```

When using the **delimiter** command, you should avoid the use of the backslash (****) character because that is the escape character for MySQL.

The following is an example of a function that takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use **delimiter** because the function definition contains no internal **;** statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50)
-> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+-----+
| hello('world') |
+-----+
| Hello, world!  |
+-----+
1 row in set (0.00 sec)
```

For information about invoking stored procedures from within programs written in a language that has a MySQL interface, see [Section 19.2.4, “CALL Statement Syntax”](#).

19.2.2. ALTER PROCEDURE and ALTER FUNCTION Syntax

```
ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]

characteristic:
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'
```

This statement can be used to change the characteristics of a stored procedure or function. You must have the **ALTER ROUTINE** priv-

ilege for the routine. (That privilege is granted automatically to the routine creator.) If binary logging is enabled, the **ALTER FUNCTION** statement might also require the **SUPER** privilege, as described in [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

More than one change may be specified in an **ALTER PROCEDURE** or **ALTER FUNCTION** statement.

19.2.3. DROP PROCEDURE and DROP FUNCTION Syntax

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

This statement is used to drop a stored procedure or function. That is, the specified routine is removed from the server. You must have the **ALTER ROUTINE** privilege for the routine. (That privilege is granted automatically to the routine creator.)

The **IF EXISTS** clause is a MySQL extension. It prevents an error from occurring if the procedure or function does not exist. A warning is produced that can be viewed with **SHOW WARNINGS**.

19.2.4. CALL Statement Syntax

```
CALL sp_name( [parameter [, ...]] )
CALL sp_name( )
```

The **CALL** statement invokes a procedure that was defined previously with **CREATE PROCEDURE**.

CALL can pass back values to its caller using parameters that are declared as **OUT** or **INOUT** parameters. It also “returns” the number of rows affected, which a client program can obtain at the SQL level by calling the **ROW_COUNT()** function and from C by calling the **mysql_affected_rows()** C API function.

As of MySQL 5.1.13, stored procedures that take no arguments now can be invoked without parentheses. That is, **CALL p()** and **CALL p** are equivalent.

To get back a value from a procedure using an **OUT** or **INOUT** parameter, pass the parameter by means of a user variable, and then check the value of the variable after the procedure returns. (If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an **IN** or **INOUT** parameter.) For an **INOUT** parameter, initialize its value before passing it to the procedure. The following procedure has an **OUT** parameter that the procedure sets to the current server version, and an **INOUT** value that the procedure increments by one from its current value:

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
  # Set value of OUT parameter
  SELECT VERSION() INTO ver_param;
  # Increment value of INOUT parameter
  SET incr_param = incr_param + 1;
END;
```

Before calling the procedure, initialize the variable to be passed as the **INOUT** parameter. After calling the procedure, the values of the two variables will have been set or modified:

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+-----+-----+
| @version | @increment |
+-----+-----+
| 5.1.12-beta-log | 11 |
+-----+-----+
```

If you write C programs that use the **CALL** SQL statement to execute stored procedures that produce result sets, you *must* set the **CLIENT_MULTI_RESULTS** flag, either explicitly, or implicitly by setting **CLIENT_MULTI_STATEMENTS** when you call **mysql_real_connect()**. This is because each such stored procedure produces multiple results: the result sets returned by statements executed within the procedure, as well as a result to indicate the call status. To process the result of a **CALL** statement, use a loop that calls **mysql_next_result()** to determine whether there are more results. For an example, see [Section 25.2.9, “C API Handling of Multiple Statement Execution”](#).

For programs written in a language that provides a MySQL interface, there is no native method for directly retrieving the results of **OUT** or **INOUT** parameters from **CALL** statements. To get the parameter values, pass user-defined variables to the procedure in the **CALL** statement and then execute a **SELECT** statement to produce a result set containing the variable values. The following example illustrates the technique (without error checking) for a stored procedure **p1** that has two **OUT** parameters.

```
mysql_query(mysql, "CALL p1(@param1, @param2)");
mysql_query(mysql, "SELECT @param1, @param2");
result = mysql_store_result(mysql);
row = mysql_fetch_row(result);
mysql_free_result(result);
```

After the preceding code executes, `row[0]` and `row[1]` contain the values of `@param1` and `@param2`, respectively.

To handle **INOUT** parameters, execute a statement prior to the **CALL** that sets the user variables to the values to be passed to the procedure.

19.2.5. **BEGIN ... END** Compound Statement Syntax

```
[begin_label:] BEGIN
[statement_list]
END [end_label]
```

BEGIN ... END syntax is used for writing compound statements, which can appear within stored routines and triggers. A compound statement can contain multiple statements, enclosed by the **BEGIN** and **END** keywords. *statement_list* represents a list of one or more statements. Each statement within *statement_list* must be terminated by a semicolon (`;`) statement delimiter. Note that *statement_list* is optional, which means that the empty compound statement (**BEGIN END**) is legal.

Use of multiple statements requires that a client is able to send statement strings containing the `;` statement delimiter. This is handled in the **mysql** command-line client with the **delimiter** command. Changing the `;` end-of-statement delimiter (for example, to `//`) allows `;` to be used in a routine body. For an example, see [Section 19.2.1, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#).

A compound statement can be labeled. *end_label* cannot be given unless *begin_label* also is present. If both are present, they must be the same.

The optional **[NOT] ATOMIC** clause is not yet supported. This means that no transactional savepoint is set at the start of the instruction block and the **BEGIN** clause used in this context has no effect on the current transaction.

19.2.6. **DECLARE** Statement Syntax

The **DECLARE** statement is used to define various items local to a routine:

- Local variables. See [Section 19.2.7, “Variables in Stored Routines”](#).
- Conditions and handlers. See [Section 19.2.8, “Conditions and Handlers”](#).
- Cursors. See [Section 19.2.9, “Cursors”](#).

The **SIGNAL** and **RESIGNAL** statements are not currently supported.

DECLARE is allowed only inside a **BEGIN ... END** compound statement and must be at its start, before any other statements.

Declarations must follow a certain order. Cursors must be declared before declaring handlers, and variables and conditions must be declared before declaring either cursors or handlers.

19.2.7. Variables in Stored Routines

You may declare and use variables within a routine.

19.2.7.1. **DECLARE** Local Variables

```
DECLARE var_name[, ...] type [DEFAULT value]
```

This statement is used to declare local variables. To provide a default value for the variable, include a **DEFAULT** clause. The value can be specified as an expression; it need not be a constant. If the **DEFAULT** clause is missing, the initial value is **NULL**.

Local variables are treated like routine parameters with respect to data type and overflow checking. See [Section 19.2.1, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#).

The scope of a local variable is within the **BEGIN ... END** block where it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

19.2.7.2. Variable **SET** Statement

```
SET var_name = expr [, var_name = expr] ...
```

The **SET** statement in stored routines is an extended version of the general **SET** statement. Referenced variables may be ones declared inside a routine, or global system variables.

The **SET** statement in stored routines is implemented as part of the pre-existing **SET** syntax. This allows an extended syntax of **SET a=x, b=y, ...** where different variable types (locally declared variables and global and session server variables) can be mixed. This also allows combinations of local variables and some options that make sense only for system variables; in that case, the options are recognized but ignored.

19.2.7.3. **SELECT ... INTO** Statement

```
SELECT col_name [, ...] INTO var_name [, ...] table_expr
```

This **SELECT** syntax stores selected columns directly into variables. Therefore, only a single row may be retrieved.

```
SELECT id,data INTO x,y FROM test.t1 LIMIT 1;
```

User variable names are not case sensitive. See [Section 8.4, “User-Defined Variables”](#).

Important

SQL variable names should not be the same as column names. If an SQL statement, such as a **SELECT ... INTO** statement, contains a reference to a column and a declared local variable with the same name, MySQL currently interprets the reference as the name of a variable. For example, in the following statement, **xname** is interpreted as a reference to the **xname** variable rather than the **xname** column:

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;

  SELECT xname,id INTO newname,xid
    FROM table1 WHERE xname = xname;
  SELECT newname;
END;
```

When this procedure is called, the **newname** variable returns the value **'bob'** regardless of the value of the **table1.xname** column.

See also [Section D.1, “Restrictions on Stored Routines and Triggers”](#).

19.2.8. Conditions and Handlers

Certain conditions may require specific handling. These conditions can relate to errors, as well as to general flow control inside a routine.

19.2.8.1. **DECLARE** Conditions

```
DECLARE condition_name CONDITION FOR condition_value
condition_value:
  SQLSTATE [VALUE] sqlstate_value
  | mysql_error_code
```

This statement specifies conditions that need specific handling. It associates a name with a specified error condition. The name can subsequently be used in a **DECLARE HANDLER** statement. See [Section 19.2.8.2, “DECLARE Handlers”](#).

A **condition_value** can be an SQLSTATE value or a MySQL error code.

19.2.8.2. DECLARE Handlers

```
DECLARE handler_type HANDLER FOR condition_value[,...] statement

handler_type:
  CONTINUE
  | EXIT
  | UNDO

condition_value:
  SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
  | mysql_error_code
```

The **DECLARE ... HANDLER** statement specifies handlers that each may deal with one or more conditions. If one of these conditions occurs, the specified *statement* is executed. *statement* can be a simple statement (for example, **SET var_name = value**), or it can be a compound statement written using **BEGIN** and **END** (see [Section 19.2.5, “BEGIN ... END Compound Statement Syntax”](#)).

For a **CONTINUE** handler, execution of the current routine continues after execution of the handler statement. For an **EXIT** handler, execution terminates for the **BEGIN ... END** compound statement in which the handler is declared. (This is true even if the condition occurs in an inner block.) The **UNDO** handler type statement is not yet supported.

If a condition occurs for which no handler has been declared, the default action is **EXIT**.

A *condition_value* can be any of the following values:

- An SQLSTATE value or a MySQL error code.
- A condition name previously specified with **DECLARE ... CONDITION**. See [Section 19.2.8.1, “DECLARE Conditions”](#).
- **SQLWARNING** is shorthand for all SQLSTATE codes that begin with **01**.
- **NOT FOUND** is shorthand for all SQLSTATE codes that begin with **02**.
- **SQLEXCEPTION** is shorthand for all SQLSTATE codes not caught by **SQLWARNING** or **NOT FOUND**.

Example:

```
mysql> CREATE TABLE test.t (s1 int,primary key (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
-> BEGIN
->   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
->   SET @x = 1;
->   INSERT INTO test.t VALUES (1);
->   SET @x = 2;
->   INSERT INTO test.t VALUES (1);
->   SET @x = 3;
-> END;
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo();//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+-----+
| @x    |
+-----+
| 3     |
+-----+
1 row in set (0.00 sec)
```

The example associates a handler with SQLSTATE 23000, which occurs for a duplicate-key error. Notice that **@x** is **3**, which shows that MySQL executed to the end of the procedure. If the line **DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;** had not been present, MySQL would have taken the default path (**EXIT**) after the second **INSERT** failed due to the

PRIMARY KEY constraint, and **SELECT @x** would have returned 2.

If you want to ignore a condition, you can declare a **CONTINUE** handler for it and associate it with an empty block. For example:

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

The statement associated with a handler cannot use **ITERATE** or **LEAVE** to refer to labels for blocks that enclose the handler declaration. That is, the scope of a block label does not include the code for handlers declared within the block. Consider the following example, where the **REPEAT** block has a label of **retry**:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  retry:
  REPEAT
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLWARNING
    BEGIN
      ITERATE retry; # illegal
    END;
  END;
  IF i < 0 THEN
    LEAVE retry;      # legal
  END IF;
  SET i = i - 1;
  UNTIL FALSE END REPEAT;
END;
```

The label is in scope for the **IF** statement within the block. It is not in scope for the **CONTINUE** handler, so the reference there is invalid and results in an error:

```
ERROR 1308 (42000): LEAVE with no matching label: retry
```

To avoid using references to outer labels in handlers, you can use different strategies:

- If you want to leave the block, you can use an **EXIT** handler:

```
DECLARE EXIT HANDLER FOR SQLWARNING BEGIN END;
```

- If you want to iterate, you can set a status variable in the handler that can be checked in the enclosing block to determine whether the handler was invoked. The following example uses the variable **done** for this purpose:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  DECLARE done INT DEFAULT FALSE;
  retry:
  REPEAT
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLWARNING
    BEGIN
      SET done = TRUE;
    END;
  END;
  IF NOT done AND i < 0 THEN
    LEAVE retry;
  END IF;
  SET i = i - 1;
  UNTIL FALSE END REPEAT;
END;
```

19.2.9. Cursors

Cursors are supported inside stored procedures and functions and triggers. The syntax is as in embedded SQL. Cursors are currently asensitive, read-only, and non-scrolling. Asensitive means that the server may or may not make a copy of its result table.

Cursors must be declared before declaring handlers, and variables and conditions must be declared before declaring either cursors or handlers.

Example:

```

CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE a CHAR(16);
  DECLARE b,c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

  OPEN cur1;
  OPEN cur2;

  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF NOT done THEN
      IF b < c THEN
        INSERT INTO test.t3 VALUES (a,b);
      ELSE
        INSERT INTO test.t3 VALUES (a,c);
      END IF;
    END IF;
  UNTIL done END REPEAT;

  CLOSE cur1;
  CLOSE cur2;
END

```

19.2.9.1. Declaring Cursors

```
DECLARE cursor_name CURSOR FOR select_statement
```

This statement declares a cursor. Multiple cursors may be declared in a routine, but each cursor in a given block must have a unique name.

The **SELECT** statement cannot have an **INTO** clause.

19.2.9.2. Cursor **OPEN** Statement

```
OPEN cursor_name
```

This statement opens a previously declared cursor.

19.2.9.3. Cursor **FETCH** Statement

```
FETCH cursor_name INTO var_name [, var_name] ...
```

This statement fetches the next row (if a row exists) using the specified open cursor, and advances the cursor pointer.

If no more rows are available, a No Data condition occurs with SQLSTATE value 02000. To detect this condition, you can set up a handler for it. An example is shown in [Section 19.2.9, “Cursors”](#).

19.2.9.4. Cursor **CLOSE** Statement

```
CLOSE cursor_name
```

This statement closes a previously opened cursor.

If not closed explicitly, a cursor is closed at the end of the compound statement in which it was declared.

19.2.10. Flow Control Constructs

The **IF**, **CASE**, **LOOP**, **WHILE**, **REPEAT**, **ITERATE**, and **LEAVE** constructs are fully implemented.

Many of these constructs contain other statements, as indicated by the grammar specifications in the following sections. Such constructs may be nested. For example, an **IF** statement might contain a **WHILE** loop, which itself contains a **CASE** statement.

FOR loops are not currently supported.

19.2.10.1. IF Statement

```
IF search_condition THEN statement_list
  [ELSEIF search_condition THEN statement_list] ...
  [ELSE statement_list]
END IF
```

IF implements a basic conditional construct. If the *search_condition* evaluates to true, the corresponding SQL statement list is executed. If no *search_condition* matches, the statement list in the **ELSE** clause is executed. Each *statement_list* consists of one or more statements.

Note

There is also an **IF()** function, which differs from the **IF statement** described here. See [Section 11.3, “Control Flow Functions”](#).

19.2.10.2. CASE Statement

```
CASE case_value
  WHEN when_value THEN statement_list
  [WHEN when_value THEN statement_list] ...
  [ELSE statement_list]
END CASE
```

Or:

```
CASE
  WHEN search_condition THEN statement_list
  [WHEN search_condition THEN statement_list] ...
  [ELSE statement_list]
END CASE
```

The **CASE** statement for stored routines implements a complex conditional construct. If a *search_condition* evaluates to true, the corresponding SQL statement list is executed. If no search condition matches, the statement list in the **ELSE** clause is executed. Each *statement_list* consists of one or more statements.

Note

The syntax of the **CASE statement** shown here for use inside stored routines differs slightly from that of the SQL **CASE expression** described in [Section 11.3, “Control Flow Functions”](#). The **CASE** statement cannot have an **ELSE NULL** clause, and it is terminated with **END CASE** instead of **END**.

19.2.10.3. LOOP Statement

```
[begin_label:] LOOP
  statement_list
END LOOP [end_label]
```

LOOP implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements. The statements within the loop are repeated until the loop is exited; usually this is accomplished with a **LEAVE** statement.

A **LOOP** statement can be labeled. *end_label* cannot be given unless *begin_label* also is present. If both are present, they must be the same.

19.2.10.4. LEAVE Statement

```
LEAVE label
```

This statement is used to exit any labeled flow control construct. It can be used within **BEGIN . . . END** or loop constructs (**LOOP**, **REPEAT**, **WHILE**).

19.2.10.5. ITERATE Statement

```
ITERATE label
```

ITERATE can appear only within **LOOP**, **REPEAT**, and **WHILE** statements. **ITERATE** means “do the loop again.”

Example:

```
CREATE PROCEDURE doitrate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END
```

19.2.10.6. REPEAT Statement

```
[begin_label:] REPEAT
  statement_list
UNTIL search_condition
END REPEAT [end_label]
```

The statement list within a **REPEAT** statement is repeated until the *search_condition* is true. Thus, a **REPEAT** always enters the loop at least once. *statement_list* consists of one or more statements.

A **REPEAT** statement can be labeled. *end_label* cannot be given unless *begin_label* also is present. If both are present, they must be the same.

Example:

```
mysql> delimiter //
mysql> CREATE PROCEDURE dorepeat(p1 INT)
-> BEGIN
->   SET @x = 0;
->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
-> END
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+-----+
| @x    |
+-----+
| 1001  |
+-----+
1 row in set (0.00 sec)
```

19.2.10.7. WHILE Statement

```
[begin_label:] WHILE search_condition DO
  statement_list
END WHILE [end_label]
```

The statement list within a **WHILE** statement is repeated as long as the *search_condition* is true. *statement_list* consists of one or more statements.

A **WHILE** statement can be labeled. *end_label* cannot be given unless *begin_label* also is present. If both are present, they must be the same.

Example:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    SET v1 = v1 - 1;
  END WHILE;
END
```

19.3. Stored Procedures, Functions, Triggers, and `LAST_INSERT_ID()`

Within the body of a stored routine (procedure or function) or a trigger, the value of `LAST_INSERT_ID()` changes the same way as for statements executed outside the body of these kinds of objects (see [Section 11.11.3, “Information Functions”](#)). The effect of a stored routine or trigger upon the value of `LAST_INSERT_ID()` that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of `LAST_INSERT_ID()`, the changed value will be seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements will not see a changed value.

19.4. Binary Logging of Stored Routines and Triggers

The binary log contains information about SQL statements that modify database contents. This information is stored in the form of “events” that describe the modifications. The binary log has two important purposes:

- For replication, the master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See [Section 15.4, “Replication Implementation”](#).
- Certain data recovery operations require use of the binary log. After a backup file has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See [Section 5.9.2.2, “Using Backups for Recovery”](#).

This section describes how MySQL 5.1 handles binary logging for stored routines (procedures and functions) and triggers. It also states the current conditions that the implementation places on the use of stored routines, and then provides additional information about the reasons for these conditions.

In general, the issues described here result when binary logging occurs at the SQL statement level. If you use row-based binary logging, the log contains changes made to individual rows as a result of executing SQL statements. For general information about row-based logging, see [Section 15.1.2, “Replication Formats”](#).

When using row-based logging, definitions of stored routines and triggers are replicated as statements. When routines or triggers execute, row changes are logged, not the statements that execute them. For stored procedures, this means that the `CALL` statement is not logged. For stored functions, row changes made within the function are logged, not the function invocation. For triggers, row changes made by the trigger are logged. On the slave side, only the row changes are seen, not the routine or trigger invocation.

Unless noted otherwise, the remarks here assume that you have enabled binary logging by starting the server with the `--log-bin` option. (See [Section 5.11.4, “The Binary Log”](#).) If the binary log is not enabled, replication is not possible, nor is the binary log available for data recovery.

The current conditions on the use of stored functions in MySQL 5.1 can be summarized as follows. These conditions do not apply to stored procedures and they do not apply unless binary logging is enabled.

- To create or alter a stored function, you must have the `SUPER` privilege, in addition to the `CREATE ROUTINE` or `ALTER ROUTINE` privilege that is normally required.
- When you create a stored function, you must declare either that it is deterministic or that it does not modify data. Otherwise, it may be unsafe for data recovery or replication.
- To relax the preceding conditions on function creation (that you must have the `SUPER` privilege and that a function must be declared deterministic or to not modify data), set the global `log_bin_trust_function_creators` system variable to 1. By default, this variable has a value of 0, but you can change it like this:

```
mysql> SET GLOBAL log_bin_trust_function_creators = 1;
```

You can also set this variable by using the `--log-bin-trust-function-creators=1` option when starting the server.

If binary logging is not enabled, `log_bin_trust_function_creators` does not apply and `SUPER` is not required for function creation.

Triggers are similar to stored functions, so the preceding remarks regarding functions also apply to triggers with the following exception: **CREATE TRIGGER** does not have an optional **DETERMINISTIC** characteristic, so triggers are assumed to be always deterministic. However, this assumption might in some cases be invalid. For example, the **UUID()** function is non-deterministic (and does not replicate). You should be careful about using such functions in triggers.

Triggers can update tables, so error messages similar to those for stored functions occur with **CREATE TRIGGER** if you do not have the **TRIGGER** privilege (**SUPER** before MySQL 5.1.6) and **log_bin_trust_function_creators** is 0. (On the slave side, the slave uses the trigger **DEFINER** attribute to determine which user is considered to be the creator of the trigger.)

The following discussion provides additional detail about the logging implementation and its implications. This discussion applies only for statement-based logging, and not for row-based logging, with the exception of the first item: **CREATE** and **DROP** statements are logged as statements regardless of the logging mode.

- The server writes **CREATE PROCEDURE**, **CREATE FUNCTION**, **ALTER PROCEDURE**, **ALTER FUNCTION**, **DROP PROCEDURE**, and **DROP FUNCTION** statements to the binary log.
- A stored function invocation is logged as a **SELECT** statement if the function changes data and occurs within a statement that would not otherwise be logged. This prevents non-replication of data changes that result from use of stored functions in non-logged statements. For example, **SELECT** statements are not written to the binary log, but a **SELECT** might invoke a stored function that makes changes. To handle this, a **SELECT func_name()** statement is written to the binary log when the given function makes a change. Suppose that the following statements are executed on the master:

```
CREATE FUNCTION f1(a INT) RETURNS INT
BEGIN
  IF (a < 3) THEN
    INSERT INTO t2 VALUES (a);
  END IF;
END;

CREATE TABLE t1 (a INT);
INSERT INTO t1 VALUES (1),(2),(3);

SELECT f1(a) FROM t1;
```

When the **SELECT** statement executes, the function **f1()** is invoked three times. Two of those invocations insert a row, and MySQL logs a **SELECT** statement for each of them. That is, MySQL writes the following statements to the binary log:

```
SELECT f1(1);
SELECT f1(2);
```

The server also logs a **SELECT** statement for a stored function invocation when the function invokes a stored procedure that causes an error. In this case, the server writes the **SELECT** statement to the log along with the expected error code. On the slave, if the same error occurs, that is the expected result and replication continues. Otherwise, replication stops.

Note: Before MySQL 5.1.7, you will see these **SELECT func_name()** statements logged as **DO func_name()**. The change to **SELECT** was made because use of **DO** was found to yield insufficient control over error code checking.

- Logging stored function invocations rather than the statements executed by a function has a security implication for replication, which arises from two factors:
 - It is possible for a function to follow different execution paths on master and slave servers.
 - Statements executed on a slave are processed by the slave SQL thread which has full privileges.

The implication is that although a user must have the **CREATE ROUTINE** privilege to create a function, the user can write a function containing a dangerous statement that will execute only on the slave where the statement is processed by the SQL thread that has full privileges. For example, if the master and slave servers have server ID values of 1 and 2, respectively, a user on the master server could create and invoke an unsafe function **unsafe_func()** as follows:

```
mysql> delimiter //
mysql> CREATE FUNCTION unsafe_func () RETURNS INT
-> BEGIN
->   IF @@server_id=2 THEN dangerous_statement; END IF;
->   RETURN 1;
-> END;
-> //
mysql> delimiter ;
mysql> INSERT INTO t VALUES(unsafe_func());
```


The `CREATE FUNCTION` and `INSERT` statements are written to the binary log, so the slave will execute them. Because the slave SQL thread has full privileges, it will execute the dangerous statement. Thus, the function invocation has different effects on the master and slave and is not replication-safe.

To guard against this danger for servers that have binary logging enabled, stored function creators must have the `SUPER` privilege, in addition to the usual `CREATE ROUTINE` privilege that is required. Similarly, to use `ALTER FUNCTION`, you must have the `SUPER` privilege in addition to the `ALTER ROUTINE` privilege. Without the `SUPER` privilege, an error will occur:

```
ERROR 1419 (HY000): You do not have the SUPER privilege and
binary logging is enabled (you *might* want to use the less safe
log_bin_trust_function_creators variable)
```

If you do not want to require function creators to have the `SUPER` privilege (for example, if all users with the `CREATE ROUTINE` privilege on your system are experienced application developers), set the global `log_bin_trust_function_creators` system variable to 1. You can also set this variable by using the `--log-bin-trust-function-creators=1` option when starting the server. If binary logging is not enabled, `log_bin_trust_function_creators` does not apply and `SUPER` is not required for function creation.

- If a function that performs updates is non-deterministic, it is not repeatable. This can have two undesirable effects:
 - It will make a slave different from the master.
 - Restored data will be different from the original data.

To deal with these problems, MySQL enforces the following requirement: On a master server, creation and alteration of a function is refused unless you declare the function to be deterministic or to not modify data. Two sets of function characteristics apply here:

- The `DETERMINISTIC` and `NOT DETERMINISTIC` characteristics indicate whether a function always produces the same result for given inputs. The default is `NOT DETERMINISTIC` if neither characteristic is given. To declare that a function is deterministic, you must specify `DETERMINISTIC` explicitly.

Use of the `NOW()` function (or its synonyms) or `RAND()` does not necessarily make a function non-deterministic. For `NOW()`, the binary log includes the timestamp and replicates correctly. `RAND()` also replicates correctly as long as it is invoked only once within a function. (You can consider the function execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

`SYSDATE()` is not affected by the timestamps in the binary log, so it causes stored routines to be non-deterministic if statement-based logging is used. This does not occur if row-based logging is used, or if the server is started with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`.

- The `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, and `MODIFIES SQL DATA` characteristics provide information about whether the function reads or writes data. Either `NO SQL` or `READS SQL DATA` indicates that a function does not change data, but you must specify one of these explicitly because the default is `CONTAINS SQL` if no characteristic is given.

By default, for a `CREATE FUNCTION` statement to be accepted, `DETERMINISTIC` or one of `NO SQL` and `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_function_creators
variable)
```

If you set `log_bin_trust_function_creators` to 1, the requirement that functions be deterministic or not modify data is dropped.

Assessment of the nature of a function is based on the “honesty” of the creator: MySQL does not check that a function declared `DETERMINISTIC` is free of statements that produce non-deterministic results.

- Stored procedure calls are logged at the statement level rather than at the `CALL` level. That is, the server does not log the `CALL` statement, it logs those statements within the procedure that actually execute. As a result, the same changes that occur on the master will be observed on slave servers. This prevents problems that could result from a procedure having different execution paths on different machines.

In general, statements executed within a stored procedure are written to the binary log using the same rules that would apply were the statements to be executed in standalone fashion. Some special care is taken when logging procedure statements because state-

ment execution within procedures is not quite the same as in non-procedure context:

- A statement to be logged might contain references to local procedure variables. These variables do not exist outside of stored procedure context, so a statement that refers to such a variable cannot be logged literally. Instead, each reference to a local variable is replaced by this construct for logging purposes:

```
NAME_CONST(var_name, var_value)
```

var_name is the local variable name, and *var_value* is a constant indicating the value that the variable has at the time the statement is logged. `NAME_CONST()` has a value of *var_value*, and a “name” of *var_name*. Thus, if you invoke this function directly, you get a result like this:

```
mysql> SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
|      14 |
+-----+
```

`NAME_CONST()` allows a logged standalone statement to be executed on a slave with the same effect as the original statement that was executed on the master within a stored procedure.

- A statement to be logged might contain references to user-defined variables. To handle this, MySQL writes a `SET` statement to the binary log to make sure that the variable exists on the slave with the same value as on the master. For example, if a statement refers to a variable `@my_var`, that statement will be preceded in the binary log by the following statement, where *value* is the value of `@my_var` on the master:

```
SET @my_var = value;
```

- Procedure calls can occur within a committed or rolled-back transaction. Previously, `CALL` statements were logged even if they occurred within a rolled-back transaction. As of MySQL 5.0.12, transactional context is accounted for so that the transactional aspects of procedure execution are replicated correctly. That is, the server logs those statements within the procedure that actually execute and modify data, and also logs `BEGIN`, `COMMIT`, and `ROLLBACK` statements as necessary. For example, if a procedure updates only transactional tables and is executed within a transaction that is rolled back, those updates are not logged. If the procedure occurs within a committed transaction, `BEGIN` and `COMMIT` statements are logged with the updates. For a procedure that executes within a rolled-back transaction, its statements are logged using the same rules that would apply if the statements were executed in standalone fashion:
 - Updates to transactional tables are not logged.
 - Updates to non-transactional tables are logged because rollback does not cancel them.
 - Updates to a mix of transactional and non-transactional tables are logged surrounded by `BEGIN` and `ROLLBACK` so that slaves will make the same changes and rollbacks as on the master.
- A stored procedure call is *not* written to the binary log at the statement level if the procedure is invoked from within a stored function. In that case, the only thing logged is the statement that invokes the function (if it occurs within a statement that is logged) or a `DO` statement (if it occurs within a statement that is not logged). For this reason, care should be exercised in the use of stored functions that invoke a procedure, even if the procedure is otherwise safe in itself.

Chapter 20. Triggers

A trigger is a named database object that is associated with a table and that is activated when a particular event occurs for the table. For example, the following statements create a table and an `INSERT` trigger. The trigger sums the values inserted into one of the table's columns:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
-> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

Important

MySQL triggers are activated by SQL statements *only*. They are not activated by changes in tables made by APIs that do not transmit SQL statements to the MySQL Server; in particular, they are not activated by updates made using the `NDB` API.

This chapter describes the syntax for creating and dropping triggers, and shows some examples of how to use them. Discussion of restrictions on use of triggers is given in [Section D.1, “Restrictions on Stored Routines and Triggers”](#). Remarks regarding binary logging as it applies to triggers are given in [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

For answers to some common questions about triggers in MySQL 5.1, see [Section A.5, “MySQL 5.1 FAQ — Triggers”](#).

20.1. CREATE TRIGGER Syntax

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt
```

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Currently, `CREATE TRIGGER` requires the `TRIGGER` privilege for the table associated with the trigger. (This statement requires the `SUPER` privilege prior to MySQL 5.1.6.)

MySQL Enterprise

For expert advice on creating triggers subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

The trigger becomes associated with the table named *tbl_name*, which must refer to a permanent table. You cannot associate a trigger with a `TEMPORARY` table or a view.

When the trigger is activated, the `DEFINER` clause determines the privileges that apply, as described later in this section.

trigger_time is the trigger action time. It can be `BEFORE` or `AFTER` to indicate that the trigger activates before or after the statement that activated it.

trigger_event indicates the kind of statement that activates the trigger. The *trigger_event* can be one of the following:

- **INSERT**: The trigger is activated whenever a new row is inserted into the table; for example, through `INSERT`, `LOAD DATA`, and `REPLACE` statements.
- **UPDATE**: The trigger is activated whenever a row is modified; for example, through `UPDATE` statements.
- **DELETE**: The trigger is activated whenever a row is deleted from the table; for example, through `DELETE` and `REPLACE` statements. However, `DROP TABLE` and `TRUNCATE` statements on the table do *not* activate this trigger, because they do not use `DELETE`. Dropping a partition does not activate `DELETE` triggers, either. See [Section 12.2.9, “TRUNCATE Syntax”](#).

It is important to understand that the *trigger_event* does not represent a literal type of SQL statement that activates the trigger so much as it represents a type of table operation. For example, an `INSERT` trigger is activated by not only `INSERT` statements but also `LOAD DATA` statements because both statements insert rows into a table.

A potentially confusing example of this is the `INSERT INTO ... ON DUPLICATE KEY UPDATE ...` syntax: a `BEFORE INSERT` trigger will activate for every row, followed by either an `AFTER INSERT` trigger or both the `BEFORE UPDATE` and `AFTER UPDATE` triggers, depending on whether there was a duplicate key for the row.

There cannot be two triggers for a given table that have the same trigger action time and event. For example, you cannot have two `BEFORE UPDATE` triggers for a table. But you can have a `BEFORE UPDATE` and a `BEFORE INSERT` trigger, or a `BEFORE UPDATE` and an `AFTER UPDATE` trigger.

`trigger_stmt` is the statement to execute when the trigger activates. If you want to execute multiple statements, use the `BEGIN ... END` compound statement construct. This also enables you to use the same statements that are allowable within stored routines. See [Section 19.2.5, “BEGIN ... END Compound Statement Syntax”](#). Some statements are not allowed in triggers; see [Section D.1, “Restrictions on Stored Routines and Triggers”](#).

MySQL stores the `sql_mode` system variable setting that is in effect at the time a trigger is created, and always executes the trigger with this setting in force, *regardless of the current server SQL mode*.

Note

Currently, triggers are not activated by cascaded foreign key actions. This limitation will be lifted as soon as possible.

In MySQL 5.1, you can write triggers containing direct references to tables by name, such as the trigger named `testref` shown in this example:

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON test1
FOR EACH ROW BEGIN
  INSERT INTO test2 SET a2 = NEW.a1;
  DELETE FROM test3 WHERE a3 = NEW.a1;
  UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
END;
|
DELIMITER ;

INSERT INTO test3 (a3) VALUES
(NULL), (NULL), (NULL), (NULL), (NULL),
(NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
(0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
```

Suppose that you insert the following values into table `test1` as shown here:

```
mysql> INSERT INTO test1 VALUES
-> (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8 Duplicates: 0 Warnings: 0
```

As a result, the data in the four tables will be as follows:

```
mysql> SELECT * FROM test1;
+-----+
| a1    |
+-----+
| 1     |
| 3     |
| 1     |
| 7     |
| 1     |
| 8     |
| 4     |
| 4     |
+-----+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test2;
+-----+
| a2    |
+-----+
```

```

+-----+
| 1 |
| 3 |
| 1 |
| 7 |
| 1 |
| 8 |
| 4 |
| 4 |
+-----+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test3;
+-----+
| a3 |
+-----+
| 2 |
| 5 |
| 6 |
| 9 |
| 10 |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM test4;
+-----+
| a4 | b4 |
+-----+
| 1 | 3 |
| 2 | 0 |
| 3 | 1 |
| 4 | 2 |
| 5 | 0 |
| 6 | 0 |
| 7 | 1 |
| 8 | 1 |
| 9 | 0 |
| 10 | 0 |
+-----+
10 rows in set (0.00 sec)

```

You can refer to columns in the subject table (the table associated with the trigger) by using the aliases `OLD` and `NEW`. `OLD.col_name` refers to a column of an existing row before it is updated or deleted. `NEW.col_name` refers to the column of a new row to be inserted or an existing row after it is updated.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at trigger activation time. If a `user` value is given, it should be a MySQL account in `'user_name'@'host_name'` format (the same format used in the `GRANT` statement). The `user_name` and `host_name` values both are required. `CURRENT_USER` also can be given as `CURRENT_USER()`. The default `DEFINER` value is the user who executes the `CREATE TRIGGER` statement. (This is the same as `DEFINER = CURRENT_USER`.)

If you specify the `DEFINER` clause, you cannot set the value to any account but your own unless you have the `SUPER` privilege. These rules determine the legal `DEFINER` user values:

- If you do not have the `SUPER` privilege, the only legal `user` value is your own account, either specified literally or by using `CURRENT_USER`. You cannot set the definer to some other account.
- If you have the `SUPER` privilege, you can specify any syntactically legal account name. If the account does not actually exist, a warning is generated.

Although it is possible to create triggers with a non-existent `DEFINER` value, it is not a good idea for such triggers to be activated until the definer actually does exist. Otherwise, the behavior with respect to privilege checking is undefined.

Note: Prior to MySQL 5.1.6, MySQL requires the `SUPER` privilege for the use of `CREATE TRIGGER`, so only the second of the preceding rules applies. As of 5.1.6, `CREATE TRIGGER` requires the `TRIGGER` privilege and `SUPER` is required only to be able to set `DEFINER` to a value other than your own account.

MySQL checks trigger privileges like this:

- At `CREATE TRIGGER` time, the user that issues the statement must have the `TRIGGER` privilege. (`SUPER` prior to MySQL 5.1.6.)
- At trigger activation time, privileges are checked against the `DEFINER` user. This user must have these privileges:
 - The `TRIGGER` privilege. (`SUPER` prior to MySQL 5.1.6.)

- The **SELECT** privilege for the subject table if references to table columns occur via *OLD.col_name* or *NEW.col_name* in the trigger definition.
- The **UPDATE** privilege for the subject table if table columns are targets of **SET NEW.col_name = value** assignments in the trigger definition.
- Whatever other privileges normally are required for the statements executed by the trigger.

20.2. DROP TRIGGER Syntax

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

This statement drops a trigger. The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. **DROP TRIGGER** was added in MySQL 5.0.2. Its use requires the **TRIGGER** privilege for the table associated with the trigger. (This statement requires the **SUPER** privilege prior to MySQL 5.1.6.)

Use **IF EXISTS** to prevent an error from occurring for a trigger that does not exist. A **NOTE** is generated for a non-existent trigger when using **IF EXISTS**. See [Section 12.5.4.32, “SHOW WARNINGS Syntax”](#). The **IF EXISTS** clause was added in MySQL 5.1.14.

Note

When upgrading from a version of MySQL older than MySQL 5.0.10 to 5.0.10 or newer — including all MySQL 5.1 releases — you must drop all triggers *before upgrading* and re-create them afterward, or else **DROP TRIGGER** does not work after the upgrade. See [Section 2.11.1, “Upgrading from MySQL 5.0 to 5.1”](#), for a suggested upgrade procedure.

20.3. Using Triggers

This section discusses how to use triggers in MySQL 5.1 and some limitations regarding their use. Additional information about trigger limitations is given in [Section D.1, “Restrictions on Stored Routines and Triggers”](#).

A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Some uses for triggers are to perform checks of values to be inserted into a table or to perform calculations on values involved in an update.

A trigger is associated with a table and is defined to activate when an **INSERT**, **DELETE**, or **UPDATE** statement for the table executes. A trigger can be set to activate either before or after the triggering statement. For example, you can have a trigger activate before each row that is deleted from a table or after each row that is updated.

Important

MySQL triggers are activated by SQL statements *only*. They are not activated by changes in tables made by APIs that do not transmit SQL statements to the MySQL Server; in particular, they are not activated by updates made using the **NDB** API.

To create a trigger or drop a trigger, use the **CREATE TRIGGER** or **DROP TRIGGER** statement. The syntax for these statements is described in [Section 20.1, “CREATE TRIGGER Syntax”](#), and [Section 20.2, “DROP TRIGGER Syntax”](#).

Here is a simple example that associates a trigger with a table for **INSERT** statements. It acts as an accumulator to sum the values inserted into one of the columns of the table.

The following statements create a table and a trigger for it:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
-> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

The **CREATE TRIGGER** statement creates a trigger named *ins_sum* that is associated with the *account* table. It also includes clauses that specify the trigger activation time, the triggering event, and what to do with the trigger activates:

- The keyword **BEFORE** indicates the trigger action time. In this case, the trigger should activate before each row inserted into the table. The other allowable keyword here is **AFTER**.

- The keyword **INSERT** indicates the event that activates the trigger. In the example, **INSERT** statements cause trigger activation. You can also create triggers for **DELETE** and **UPDATE** statements.
- The statement following **FOR EACH ROW** defines the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering statement. In the example, the triggered statement is a simple **SET** that accumulates the values inserted into the **amount** column. The statement refers to the column as **NEW.amount** which means “the value of the **amount** column to be inserted into the new row.”

To use the trigger, set the accumulator variable to zero, execute an **INSERT** statement, and then see what value the variable has afterward:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----+
| Total amount inserted |
+-----+
| 1852.48                |
+-----+
```

In this case, the value of **@sum** after the **INSERT** statement has executed is **14.98 + 1937.50 - 100**, or **1852.48**.

To destroy the trigger, use a **DROP TRIGGER** statement. You must specify the schema name if the trigger is not in the default schema:

```
mysql> DROP TRIGGER test.ins_sum;
```

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

In addition to the requirement that trigger names be unique for a schema, there are other limitations on the types of triggers you can create. In particular, you cannot have two triggers for a table that have the same activation time and activation event. For example, you cannot define two **BEFORE INSERT** triggers or two **AFTER UPDATE** triggers for a table. This should rarely be a significant limitation, because it is possible to define a trigger that executes multiple statements by using the **BEGIN . . . END** compound statement construct after **FOR EACH ROW**. (An example appears later in this section.)

The **OLD** and **NEW** keywords enable you to access columns in the rows affected by a trigger. (**OLD** and **NEW** are not case sensitive.) In an **INSERT** trigger, only **NEW.col_name** can be used; there is no old row. In a **DELETE** trigger, only **OLD.col_name** can be used; there is no new row. In an **UPDATE** trigger, you can use **OLD.col_name** to refer to the columns of a row before it is updated and **NEW.col_name** to refer to the columns of the row after it is updated.

A column named with **OLD** is read-only. You can refer to it (if you have the **SELECT** privilege), but not modify it. A column named with **NEW** can be referred to if you have the **SELECT** privilege for it. In a **BEFORE** trigger, you can also change its value with **SET NEW.col_name = value** if you have the **UPDATE** privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or that are used to update a row.

In a **BEFORE** trigger, the **NEW** value for an **AUTO_INCREMENT** column is 0, not the automatically generated sequence number that will be generated when the new record actually is inserted.

OLD and **NEW** are MySQL extensions to triggers.

By using the **BEGIN . . . END** construct, you can define a trigger that executes multiple statements. Within the **BEGIN** block, you also can use other syntax that is allowed within stored routines such as conditionals and loops. However, just as for stored routines, if you use the **mysql** program to define a trigger that executes multiple statements, it is necessary to redefine the **mysql** statement delimiter so that you can use the **;** statement delimiter within the trigger definition. The following example illustrates these points. It defines an **UPDATE** trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a **BEFORE** trigger because the value needs to be checked before it is used to update the row:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
-> FOR EACH ROW
-> BEGIN
-> IF NEW.amount < 0 THEN
-> SET NEW.amount = 0;
-> ELSEIF NEW.amount > 100 THEN
-> SET NEW.amount = 100;
-> END IF;
-> END; //
mysql> delimiter ;
```

It can be easier to define a stored procedure separately and then invoke it from the trigger using a simple `CALL` statement. This is also advantageous if you want to invoke the same routine from within several triggers.

There are some limitations on what can appear in statements that a trigger executes when activated:

- The trigger cannot use the `CALL` statement to invoke stored procedures that return data to the client or that use dynamic SQL. (Stored procedures are allowed to return data to the trigger through `OUT` or `INOUT` parameters.)
- The trigger cannot use statements that explicitly or implicitly begin or end a transaction such as `START TRANSACTION`, `COMMIT`, or `ROLLBACK`.

MySQL handles errors during trigger execution as follows:

- If a `BEFORE` trigger fails, the operation on the corresponding row is not performed.
- A `BEFORE` trigger is activated by the *attempt* to insert or modify the row, regardless of whether the attempt subsequently succeeds.
- An `AFTER` trigger is executed only if the `BEFORE` trigger (if any) and the row operation both execute successfully.
- An error during either a `BEFORE` or `AFTER` trigger results in failure of the entire statement that caused trigger invocation.
- For transactional tables, failure of a statement should cause rollback of all changes performed by the statement. Failure of a trigger causes the statement to fail, so trigger failure also causes rollback. For non-transactional tables, such rollback cannot be done, so although the statement fails, any changes performed prior to the point of the error remain in effect.

Chapter 21. Event Scheduler

This chapter describes the *MySQL Event Scheduler*, for which support was added in MySQL 5.1.6, and is divided into the following sections:

- [Section 21.1, “Event Scheduler Overview”](#), provides an introduction to and conceptual overview of MySQL Events.
- [Section 21.2, “Event Scheduler Syntax”](#), discusses the SQL commands introduced in MySQL 5.1.6 for creating, altering, and dropping MySQL Events.
- [Section 21.3, “Event Metadata”](#), shows how to obtain information about events and how this information is stored by the MySQL Server.
- [Section 21.5, “The Event Scheduler and MySQL Privileges”](#), discusses the privileges required to work with events and the ramifications that events have with regard to privileges when executing.
- [Section 21.6, “Event Scheduler Limitations and Restrictions”](#), describes the restrictions and limitations of MySQL's Event Scheduler implementation.

Additional Resources. You may find the [MySQL Event Scheduler User Forum](#) of use when working with scheduled events. Here you can discuss the MySQL Event Scheduler with other MySQL users and the MySQL developers.

21.1. Event Scheduler Overview

MySQL Events are tasks that run according to a schedule. Therefore, we sometimes refer to them as *scheduled* events. When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time. Conceptually, this is similar to the idea of the Unix [crontab](#) (also known as a “cron job”) or the Windows Task Scheduler.

Scheduled tasks of this type are also sometimes known as “temporal triggers”, implying that these are objects that are triggered by the passage of time. While this is essentially correct, we prefer to use the term *events* in order to avoid confusion with triggers of the type discussed in [Chapter 20, Triggers](#). Events should more specifically not be confused with “temporary triggers”. Whereas a trigger is a database object whose statements are executed in response to a specific type of event that occurs on a given table, a (scheduled) event is an object whose statements are executed in response to the passage of a specified time interval.

While there is no provision in the SQL Standard for event scheduling, there are precedents in other database systems, and you may notice some similarities between these implementations and that found in the MySQL Server.

MySQL Events have the following major features and properties:

- In MySQL 5.1.12 and later, an event is uniquely identified by its name and the schema to which it is assigned. (Previously, an event was also unique to its definer.)
- An event performs a specific action according to a schedule. This action consists of an SQL statement, which can be a compound statement in a [BEGIN . . . END](#) block if desired (see [Section 19.2.5, “BEGIN . . . END Compound Statement Syntax”](#)). An event's timing can be either *one-time* or *recurrent*. A one-time event executes one time only. A recurrent event repeats its action at a regular interval, and the schedule for a recurring event can be assigned a specific start day and time, end day and time, both, or neither. (By default, a recurring event's schedule begins as soon as it is created, and continues indefinitely, until it is disabled or dropped.)
- Users can create, modify, and drop scheduled events using SQL statements intended for these purposes. Syntactically invalid event creation and modification statements fail with an appropriate error message. *A user may include statements in an event's action which require privileges that the user does not actually have.* The event creation or modification statement succeeds but the event's action fails. See [Section 21.5, “The Event Scheduler and MySQL Privileges”](#) for details.
- Many of the properties of an event can be set or modified using SQL statements. These properties include the event's name, timing, persistence (that is, whether it is preserved following the expiration of its schedule), status (enabled or disabled), action to be performed, and the schema to which it is assigned. See [Section 21.2.1, “ALTER EVENT Syntax”](#).

The definer of an event is the user who created the event, unless the event has been altered, in which case the definer is the user who issued the last [ALTER EVENT](#) statement effecting that event. An event can be modified by any user having the [EVENT](#) privilege on

the database for which the event is defined. (Prior to MySQL 5.1.12, only an event's definer, or a user having privileges on the `mysql.event` table, could modify a given event.) See [Section 21.5, “The Event Scheduler and MySQL Privileges”](#).

- An event's action statement may include most SQL statements permitted within stored routines.

Events are executed by a special *event scheduler thread*; when we refer to the Event Scheduler, we actually refer to this thread. When running, the event scheduler thread and its current state can be seen by users having the `SUPER` privilege in the output of `SHOW PROCESSLIST`, as shown in the discussion that follows.

The global variable `event_scheduler` determines whether the Event Scheduler is enabled and running on the server. Beginning with MySQL 5.1.12, it has one of these 3 values, which affect event scheduling as described here:

- **OFF**: The Event Scheduler is stopped. The event scheduler thread does not run, is not shown in the output of `SHOW PROCESSLIST`, and no scheduled events are executed. **OFF** is the default value for `event_scheduler`.

When the Event Scheduler is stopped (`event_scheduler` is **OFF**), it can be started by setting the value of `event_scheduler` to **ON**. (See next item.)

- **ON**: The Event Scheduler is started; the event scheduler thread runs and executes all scheduled events.

When the Event Scheduler is **ON**, the event scheduler thread is listed in the output of `SHOW PROCESSLIST` as a daemon process, and its state is represented as shown here:

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
  Id: 1
  User: root
  Host: localhost
  db: NULL
  Command: Query
  Time: 0
  State: NULL
  Info: show processlist
***** 2. row *****
  Id: 2
  User: event_scheduler
  Host: localhost
  db: NULL
  Command: Daemon
  Time: 3
  State: Waiting for next activation
  Info: NULL
2 rows in set (0.00 sec)
```

Event scheduling can be stopped by setting the value of `event_scheduler` to **OFF**.

- **DISABLED**: This value renders the Event Scheduler non-operational. When the Event Scheduler is **DISABLED**, the event scheduler thread does not run (and so does not appear in the output of `SHOW PROCESSLIST`).

When the server is running `event_scheduler` can be toggled between **ON** and **OFF** (using `SET`). It is also possible to use `0` for **OFF**, and `1` for **ON** when setting this variable. Thus, any of the following 4 statements can be used in the `mysql` client to turn on the Event Scheduler:

```
SET GLOBAL event_scheduler = ON;
SET @@global.event_scheduler = ON;
SET GLOBAL event_scheduler = 1;
SET @@global.event_scheduler = 1;
```

Similarly, any of these 4 statements can be used to turn off the Event Scheduler:

```
SET GLOBAL event_scheduler = OFF;
SET @@global.event_scheduler = OFF;
SET GLOBAL event_scheduler = 0;
SET @@global.event_scheduler = 0;
```

Although **ON** and **OFF** have numeric equivalents, the value displayed for `event_scheduler` by `SELECT` or `SHOW VARIABLES` is always one of **OFF**, **ON**, or **DISABLED**. **DISABLED** has no numeric equivalent. For this reason, **ON** and **OFF** are usually preferred over `1` and `0` when setting this variable.

Note that attempting to set `event_scheduler` without specifying it as a global variable causes an error:

```
mysql> SET @@event_scheduler = OFF;
ERROR 1229 (HY000): VARIABLE 'EVENT_SCHEDULER' IS A GLOBAL
VARIABLE AND SHOULD BE SET WITH SET GLOBAL
```

Important

It is not possible to enable or disable the Event Scheduler when the server is running. That is, you can change the value of `event_scheduler` to `DISABLED` — or from `DISABLED` to one of the other permitted values for this option — only when the server is stopped. Attempting to do so when the server is running fails with an error.

To disable the event scheduler, use one of the following two methods:

- As a command-line option when starting the server:

```
--event-scheduler=DISABLED
```

- In the server configuration file (`my.cnf`, or `my.ini` on Windows systems), include the line where it will be read by the server (for example, in a `[mysqld]` section):

```
event_scheduler=DISABLED
```

To enable the Event Scheduler, restart the server without the `--event-scheduler=DISABLED` command line option, or after removing or commenting out the line containing `event_scheduler=DISABLED` in the server configuration file, as appropriate. Alternatively, you can use `ON` (or `1`) or `OFF` (or `0`) in place of the `DISABLED` value when starting the server.

Note

You can issue event-manipulation statements when `event_scheduler` is set to `DISABLED`. No warnings or errors are generated in such cases (provided that the statements are themselves valid). However, scheduled events cannot execute until this variable is set to `ON` (or `1`). Once this has been done, the event scheduler thread executes all events whose scheduling conditions are satisfied.

In MySQL 5.1.11, `event_scheduler` behaved as follows: this variable could take one of the values `0` (or `OFF`), `1` (or `ON`), or `2`. Setting it to `0` turned event scheduling off, so that the event scheduler thread did not run; the `event_scheduler` variable could not be set to this value while the server was running. Setting it to `1` so that the event scheduler thread ran and executed scheduled events. In this state, the event scheduler thread appeared to be sleeping when viewed with `SHOW PROCESSLIST`. When `event_scheduler` was set to `2` (which was the default value), the Event Scheduler was considered to be “suspended”; the event scheduler thread ran and could be seen in the output of `SHOW PROCESSLIST` (where `Suspended` was displayed in the `State` column), but did not execute any scheduled events. The value of `event_scheduler` could be changed only between `1` (or `ON`) and `2` while the server was running. Setting it to `0` (or `OFF`) required a server restart, as did changing its value from `0` (or `OFF`) to `1` (or `ON`) or `2`.

Prior to MySQL 5.1.11, `event_scheduler` could take one of only the 2 values `0|OFF` or `1|ON`, and the default value was `0|OFF`. It was also possible to start and stop the event scheduler thread while the MySQL server was running.

For more information concerning the reasons for these changes in behaviour, see [Bug#17619](#).

Beginning with MySQL 5.1.17, starting the MySQL server with the `--skip-grant-tables` option causes `event_scheduler` to be set to `DISABLED`, overriding any other value set either on the command line or in the `my.cnf` or `my.ini` file ([Bug#26807](#)).

For SQL statements used to create, alter, and drop events, see [Section 21.2, “Event Scheduler Syntax”](#).

MySQL 5.1.6 and later provides an `EVENTS` table in the `INFORMATION_SCHEMA` database. This table can be queried to obtain information about scheduled events which have been defined on the server. See [Section 21.3, “Event Metadata”](#), and [Section 23.20, “The INFORMATION_SCHEMA EVENTS Table”](#), for more information.

For information regarding event scheduling and the MySQL privilege system, see [Section 21.5, “The Event Scheduler and MySQL Privileges”](#).

21.2. Event Scheduler Syntax

MySQL 5.1.6 and later provides several SQL statements for working with scheduled events:

- New events are defined using the `CREATE EVENT` statement. See [Section 21.2.2, “CREATE EVENT Syntax”](#).
- The definition of an existing event can be changed by means of the `ALTER EVENT` statement. See [Section 21.2.1, “ALTER EVENT Syntax”](#).
- When a scheduled event is no longer wanted or needed, it can be deleted from the server by its definer using the `DROP EVENT` statement. See [Section 21.2.3, “DROP EVENT Syntax”](#). (Whether an event persists past the end of its schedule also depends on its `ON COMPLETION` clause, if it has one. See [Section 21.2.2, “CREATE EVENT Syntax”](#).)

An event can be deleted by any user having the `EVENT` privilege for the database on which the event is defined. Prior to MySQL 5.1.2, a user other than the definer required privileges on the `mysql.event` table. See [Section 21.5, “The Event Scheduler and MySQL Privileges”](#).

21.2.1. ALTER EVENT Syntax

```
ALTER EVENT
[DEFINER = { user | CURRENT_USER }]
event_name
[ON SCHEDULE schedule]
[RENAME TO new_event_name]
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE] [DISABLE ON SLAVE]
[COMMENT 'comment']
[DO sql_statement]
```

The `ALTER EVENT` statement is used to change one or more of the characteristics of an existing event without the need to drop and re-create it. The syntax for each of the `DEFINER`, `ON SCHEDULE`, `ON COMPLETION`, `COMMENT`, `ENABLE / DISABLE`, and `DO` clauses is exactly the same as when used with `CREATE EVENT`. (See [Section 21.2.2, “CREATE EVENT Syntax”](#).)

Support for the `DEFINER` clause was added in MySQL 5.1.17.

Beginning with MySQL 5.1.12, any user can alter an event defined on a database for which that user has the `EVENT` privilege. When a user executes a successful `ALTER EVENT` statement, that user becomes the definer for the affected event.

(In MySQL 5.1.11 and earlier, an event could be altered only by its definer, or by a user having the `SUPER` privilege.)

`ALTER EVENT` works only with an existing event:

```
mysql> ALTER EVENT no_such_event
> ON SCHEDULE
> EVERY '2:3' DAY_HOUR;
ERROR 1517 (HY000): UNKNOWN EVENT 'NO_SUCH_EVENT'
```

In each of the following examples, assume that the event named `myevent` is defined as shown here:

```
CREATE EVENT myevent
ON SCHEDULE
EVERY 6 HOUR
COMMENT 'A sample comment.'
DO
UPDATE myschema.mytable SET mycol = mycol + 1;
```

The following statement changes the schedule for `myevent` from once every six hours starting immediately to once every twelve hours, starting four hours from the time the statement is run:

```
ALTER EVENT myevent
ON SCHEDULE
EVERY 12 HOUR
STARTS CURRENT_TIMESTAMP + 4 HOUR;
```

To disable `myevent`, use this `ALTER EVENT` statement:

```
ALTER EVENT myevent
DISABLE;
```

The **ON SCHEDULE** clause may use expressions involving built-in MySQL functions and user variables to obtain any of the *timestamp* or *interval* values which it contains. You may not use stored routines or user-defined functions in such expressions, nor may you use any table references; however, you may use **SELECT FROM DUAL**. This is true for both **ALTER EVENT** and **CREATE EVENT** statements. Beginning with MySQL 5.1.13, references to stored routines, user-defined functions, and tables in such cases are specifically disallowed, and fail with an error (see [Bug#22830](#)).

An **ALTER EVENT** statement that contains another **ALTER EVENT** statement in its **DO** clause appears to succeed; however, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

It is possible to change multiple characteristics of an event in a single statement. This example changes the SQL statement executed by *myevent* to one that deletes all records from *mytable*; it also changes the schedule for the event such that it executes once, one day after this **ALTER EVENT** statement is run.

```
ALTER TABLE myevent
ON SCHEDULE
  AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
DO
  TRUNCATE TABLE myschema.mytable;
```

To rename an event, use the **ALTER EVENT** statement's **RENAME TO** clause, as shown here:

```
ALTER EVENT myevent
RENAME TO yourevent;
```

The previous statement renames the event *myevent* to *yourevent*.

Note

There is no **RENAME EVENT** statement.

You can also move an event to a different database using **ALTER EVENT ... RENAME TO ...** and *db_name.table_name* notation, as shown here:

```
ALTER EVENT olddb.myevent
RENAME TO newdb.myevent;
```

In order to execute the previous statement, the user executing it must have the **EVENT** privilege on both the *olddb* and *newdb* databases.

Beginning with MySQL 5.1.18, a third value may also appear in place of **ENABLED** or **DISABLED**; **DISABLE ON SLAVE** is used on a replication slave to indicate an event which was created on the master and replicated to the slave, but which is not executed on the slave. Normally, **DISABLE ON SLAVE** is set automatically as required; however, there are some circumstances under which you may want or need to change it manually. See [Section 15.3.1.5, “Replication of Invoked Features”](#), for more information.

It is necessary to include only those options in an **ALTER EVENT** statement which correspond to characteristics that you actually wish to change; options which are omitted retain their existing values. This includes any default values for **CREATE EVENT** such as **ENABLE**.

21.2.2. CREATE EVENT Syntax

```
CREATE
  [DEFINER = { user | CURRENT_USER }]
  EVENT
  [IF NOT EXISTS]
  event_name
  ON SCHEDULE schedule
  [ON COMPLETION [NOT] PRESERVE]
  [ENABLE | DISABLE | DISABLE ON SLAVE]
  [COMMENT 'comment']
  DO sql_statement;

schedule:
  AT timestamp [+ INTERVAL interval]
  | EVERY interval
  [STARTS timestamp [+ INTERVAL interval]]
  [ENDS timestamp [+ INTERVAL interval]]

interval:
  quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
             WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
             DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

This statement creates and schedules a new event. The minimum requirements for a valid **CREATE EVENT** statement are as follows:

- The keywords **CREATE EVENT** plus an event name, which uniquely identifies the event in the current schema.
(Prior to MySQL 5.1.12, the event name needed to be unique only among events created by the same user on a given database.)
- An **ON SCHEDULE** clause, which determines when and how often the event executes.
- A **DO** clause, which contains the SQL statement to be executed by an event.

This is an example of a minimal **CREATE EVENT** statement:

```
CREATE EVENT myevent
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
  DO
    UPDATE myschema.mytable SET mycol = mycol + 1;
```

The previous statement creates an event named **myevent**. This event executes once — one hour following its creation — by running an SQL statement that increments the value of the **myschema.mytable** table's **mycol** column by 1.

The *event_name* must be a valid MySQL identifier with a maximum length of 64 characters. It may be delimited using back ticks, and may be qualified with the name of a database schema. An event is associated with both a MySQL user (the definer) and a schema, and its name must be unique among names of events within that schema. In general, the rules governing event names are the same as those for names of stored routines. See [Section 8.2, “Identifiers”](#).

If no schema is indicated as part of *event_name*, then the default (current) schema is assumed. The definer is always the current MySQL user.

(Prior to MySQL 5.1.12, it was possible for two different users to create different events having the same name on the same database schema.)

Note

MySQL uses case-insensitive comparisons when checking for the uniqueness of event names. This means that, for example, you cannot have two events named **myevent** and **MyEvent** in the same database schema.

The **DEFINER** clause specifies the MySQL account to be used when checking access privileges at trigger execution time. If a *user* value is given, it should be a MySQL account in '*user_name*'@'*host_name*' format (the same format used in the **GRANT** statement). The *user_name* and *host_name* values both are required. **CURRENT_USER** also can be given as **CURRENT_USER()**. The default **DEFINER** value is the user who executes the **CREATE TRIGGER** statement. (This is the same as **DEFINER = CURRENT_USER**.)

The **DEFINER** clause was added in MySQL 5.1.17.

IF NOT EXISTS functions in the same fashion with **CREATE EVENT** as it does when used with a **CREATE TABLE** statement; if an event named *event_name* already exists in the same schema, no action is taken, and no error results. (However, a warning is generated in such cases.)

The **ON SCHEDULE** clause determines when, how often, and for how long the *sql_statement* defined for the event repeats. This clause takes one of two forms:

- **AT timestamp** is used for a one-time event. It specifies that the event executes one time only at the date and time, given as the *timestamp*, which must include both the date and time, or must be an expression that resolves to a datetime value. You may use a value which is of either the **DATETIME** or **TIMESTAMP** type for this purpose. The *timestamp* must also be in the future — you cannot schedule an event to take place in the past. Trying to do so fails with an error, as shown here:

```
mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 2006-02-10 23:59:01 |
+-----+
1 row in set (0.04 sec)

mysql> CREATE EVENT e_totals
```

```
-> ON SCHEDULE AT '2006-02-10 23:59:00'  
-> DO INSERT INTO test.totals VALUES (NOW());  
ERROR 1522 (HY000): ACTIVATION (AT) TIME IS IN THE PAST
```

CREATE EVENT statements which are themselves invalid — for whatever reason — fail with an error.

You may use **CURRENT_TIMESTAMP** to specify the current date and time. In such a case, the event acts as soon as it is created.

In order to create an event which occurs at some point in the future relative to the current date and time — such as that expressed by the phrase “three weeks from now” — you can use the optional clause **+ INTERVAL interval**. The *interval* portion consists of two parts, a quantity and a unit of time, and follows the same syntax rules that govern intervals used in the **DATE_ADD()** function (see [Section 11.6, “Date and Time Functions”](#)). The units keywords are also the same, except that you cannot use any units involving microseconds when defining an event.

You can also combine intervals. For example, **AT CURRENT_TIMESTAMP + INTERVAL 3 WEEK + INTERVAL 2 DAY** is equivalent to “three weeks and two days from now”. Each portion of such a clause must begin with **+ INTERVAL**.

- For actions which are to be repeated at a regular interval, you can use an **EVERY** clause. The **EVERY** keyword is followed by an *interval* as described in the previous discussion of the **AT** keyword. (**+ INTERVAL** is *not* used with **EVERY**.) For example, **EVERY 6 WEEK** means “every six weeks”.

It is not possible to combine **+ INTERVAL** clauses in a single **EVERY** clause; however, you can use the same complex time units allowed in a **+ INTERVAL**. For example, “every two minutes and ten seconds” can be expressed as **EVERY '2:10' MINUTE_SECOND**.

An **EVERY** clause may also contain an optional **STARTS** clause. **STARTS** is followed by a *timestamp* value which indicates when the action should begin repeating, and may also use **+ INTERVAL interval** in order to specify an amount of time “from now”. For example, **EVERY 3 MONTH STARTS CURRENT_TIMESTAMP + 1 WEEK** means “every three months, beginning one week from now”. Similarly, you can express “every two weeks, beginning six hours and fifteen minutes from now” as **EVERY 2 WEEK STARTS CURRENT_TIMESTAMP + INTERVAL '6:15' HOUR_MINUTE**. Not specifying **STARTS** is the same as using **STARTS CURRENT_TIMESTAMP** — that is, the action specified for the event begins repeating immediately upon creation of the event.

An **EVERY** clause may also contain an optional **ENDS** clause. The **ENDS** keyword is followed by a *timestamp* value which tells MySQL when the event should stop repeating. You may also use **+ INTERVAL interval** with **ENDS**; for instance, **EVERY 12 HOUR STARTS CURRENT_TIMESTAMP + INTERVAL 30 MINUTE ENDS CURRENT_TIMESTAMP + INTERVAL 4 WEEK** is equivalent to “every twelve hours, beginning thirty minutes from now, and ending four weeks from now”. Not using **ENDS** means that the event continues executing indefinitely.

ENDS supports the same syntax for complex time units as **STARTS** does.

You may use **STARTS**, **ENDS**, both, or neither in an **EVERY** clause.

Note

Beginning with MySQL 5.1.17, **STARTS** or **ENDS** uses the MySQL server's local time zone, as shown in the **INFORMATION_SCHEMA.EVENTS** and **mysql.event** tables, as well as in the output of **SHOW EVENTS**. Previously, this information was stored using UTC ([Bug#16420](#)).

Due to this change, the **mysql.event** table must be updated before events created in earlier releases can be created, altered, viewed, or used in MySQL 5.1.17 or later.

See [Section 23.20, “The INFORMATION_SCHEMA EVENTS Table”](#), and [Section 12.5.4.16, “SHOW EVENTS”](#) for information about columns added in MySQL 5.1.17 to accommodate these changes.

The **ON SCHEDULE** clause may use expressions involving built-in MySQL functions and user variables to obtain any of the *timestamp* or *interval* values which it contains. You may not use stored routines or user-defined functions in such expressions, nor may you use any table references; however, you may use **SELECT FROM DUAL**. This is true for both **CREATE EVENT** and **ALTER EVENT** statements. Beginning with MySQL 5.1.13, references to stored routines, user-defined functions, and tables in such cases are specifically disallowed, and fail with an error (see [Bug#22830](#)).

Normally, once an event has expired, it is immediately dropped. You can override this behavior by specifying **ON COMPLETION PRESERVE**. Using **ON COMPLETION NOT PRESERVE** merely makes the default non-persistent behavior explicit.

You can create an event but keep it from being active using the `DISABLE` keyword. Alternatively, you may use `ENABLE` to make explicit the default status, which is active. This is most useful in conjunction with `ALTER EVENT` (see [Section 21.2.1, “ALTER EVENT Syntax”](#)).

Beginning with MySQL 5.1.18, a third value may also appear in place of `ENABLED` or `DISABLED`; `DISABLE ON SLAVE` is set for the status of an event on a replication slave to indicate that the event was created on the master and replicated to the slave, but is not executed on the slave. See [Section 15.3.1.5, “Replication of Invoked Features”](#).

You may supply a comment for an event using a `COMMENT` clause. *comment* may be any string of up to 64 characters that you wish to use for describing the event. The comment text, being a string literal, must be surrounded by quotation marks.

The `DO` clause specifies an action carried by the event, and consists of an SQL statement. Nearly any valid MySQL statement which can be used in a stored routine can also be used as the action statement for a scheduled event. (See [Section D.1, “Restrictions on Stored Routines and Triggers”](#).) For example, the following event `e_hourly` deletes all rows from the `sessions` table once per hour, where this table is part of the `site_activity` schema:

```
CREATE EVENT e_hourly
ON SCHEDULE
  EVERY 1 HOUR
COMMENT 'Clears out sessions table each hour.'
DO
  DELETE FROM site_activity.sessions;
```

MySQL stores the `sql_mode` system variable setting that is in effect at the time an event is created, and always executes the event with this setting in force, *regardless of the current server SQL mode*.

A `CREATE EVENT` statement that contains an `ALTER EVENT` statement in its `DO` clause appears to succeed; however, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

Note

The `SHOW` statement and `SELECT` statements that merely return a result set have no effect when used in an event; the output from these is not sent to the MySQL Monitor, nor is it stored anywhere. However, you can use statements such as `SELECT INTO` and `INSERT ... SELECT` that store a result. (See the next example in this section for an instance of the latter.)

Any reference in the `DO` clause to a table in other than the same database schema to which the event belongs must be qualified with the name of the schema in which the table occurs. (In MySQL 5.1.6, all tables referenced in event `DO` clauses had to include a reference to the database.)

As with stored routines, you can use multiple statements in the `DO` clause by bracketing them with the `BEGIN` and `END` keywords, as shown here:

```
DELIMITER |
CREATE EVENT e_daily
ON SCHEDULE
  EVERY 1 DAY
COMMENT 'Saves total number of sessions then clears the table each day.'
DO
  BEGIN
    INSERT INTO site_activity.totals (when, total)
      SELECT CURRENT_TIMESTAMP, COUNT(*)
        FROM site_activity.sessions;
    DELETE FROM site_activity.sessions;
  END |
DELIMITER ;
```

Note the use of the `DELIMITER` statement to change the statement delimiter, as with stored routines. See [Section 19.2.1, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#).

More complex compound statements, such as those used in stored routines, are possible in an event. This example uses local variables, an error handler, and a flow control construct:

```
DELIMITER |
CREATE EVENT e
ON SCHEDULE
  EVERY 5 SECOND
DO
  BEGIN
```



```
DECLARE v INTEGER;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END;

SET v = 0;

WHILE v < 5 DO
    INSERT INTO t1 VALUES (0);
    UPDATE t2 SET s1 = s1 + 1;
    SET v = v + 1;
END WHILE;
END |
DELIMITER ;
```

There is no way to pass parameters directly to or from events; however, it is possible to invoke a stored routine with parameters:

```
CREATE EVENT e_call_myproc
ON SCHEDULE
AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
DO CALL myproc(5, 27);
```

In addition, if the event's definer has the [SUPER](#) privilege, that event may read and write global variables. As granting this privilege entails a potential for abuse, extreme care must be taken in doing so.

Generally, any statements which are valid in stored routines may be used for action statements executed by events. For more information about statements allowable within stored routines, see [Section 19.2, “Stored Routine Syntax”](#). You can create an event as part of a stored routine, but an event cannot be created by another event.

21.2.3. DROP EVENT Syntax

```
DROP EVENT [IF EXISTS] event_name
```

This statement drops the event named *event_name*. The event immediately ceases being active, and is deleted completely from the server.

If the event does not exist, the error **ERROR 1517 (HY000): UNKNOWN EVENT 'EVENT_NAME'** results. You can override this and cause the statement to fail silently by using **IF EXISTS**.

Beginning with MySQL 5.1.12, an event can be dropped by any user having the [EVENT](#) privilege on the database schema to which the event to be dropped belongs. (In MySQL 5.1.11 and earlier, an event could be dropped only by its definer, or by a user having the [SUPER](#) privilege.)

21.3. Event Metadata

Information about events can be obtained as follows:

- Querying the [EVENTS](#) table of the [INFORMATION_SCHEMA](#) database. See [Section 23.20, “The INFORMATION_SCHEMA EVENTS Table”](#).
- Using the [SHOW EVENTS](#) statement. See [Section 12.5.4.16, “SHOW EVENTS”](#).
- Using the [SHOW CREATE EVENT](#) statement. See [Section 12.5.4.7, “SHOW CREATE EVENT”](#).
- A record of events executed on the server can be read from the MySQL Server's error log (see [Section 21.5, “The Event Scheduler and MySQL Privileges”](#) for an example).

21.4. Event Scheduler Status

Information about the state of the Event Scheduler for debugging and troubleshooting purposes can be obtained as follows:

- In MySQL 5.1.11 **-debug** builds, you can use the [SHOW SCHEDULER STATUS](#) statement; see [Section 12.5.4.26, “SHOW SCHEDULER STATUS Syntax”](#).

Important

This statement was removed in MySQL 5.1.12. We intend to implement an SQL statement providing similar functionality in a future MySQL release.

- Beginning with MySQL 5.1.12, event scheduler status information can be obtained by running `mysqladmin debug` (see [Section 7.10, “mysqladmin — Client for Administering a MySQL Server”](#)); after running this command, the error log contains output relating to the Event Scheduler, similar to what is shown here:

```
Events status:
LLA = Last Locked At   LUA = Last Unlocked At
WOC = Waiting On Condition  DL = Data Locked

Event scheduler status:
State      : INITIALIZED
Thread id  : 0
LLA        : init_scheduler:313
LUA        : init_scheduler:318
WOC        : NO
Workers    : 0
Executed   : 0
Data locked: NO

Event queue status:
Element count : 1
Data locked   : NO
Attempting lock : NO
LLA          : init_queue:148
LUA          : init_queue:168
WOC          : NO
Next activation : 0000-00-00 00:00:00
```

21.5. The Event Scheduler and MySQL Privileges

To enable or disable the execution of scheduled events, it is necessary to set the value of the global `event_scheduler` variable. This requires the `SUPER` privilege.

MySQL 5.1.6 introduces a privilege governing the creation, modification, and deletion of events, the `EVENT` privilege. This privilege can be bestowed using `GRANT`. For example, this `GRANT` statement confers the `EVENT` privilege for the schema named `myschema` on the user `jon@ghidora`:

```
GRANT EVENT ON myschema.* TO jon@ghidora;
```

(We assume that this user account already exists, and that we wish for it to remain unchanged otherwise.)

To grant this same user the `EVENT` privilege on all schemas would require the following statement:

```
GRANT EVENT ON *.* TO jon@ghidora;
```

The `EVENT` privilege has schema-level scope. Therefore, trying to grant it on a single table results in an error as shown:

```
mysql> GRANT EVENT ON myschema.mytable TO jon@ghidora;
ERROR 1144 (42000): ILLEGAL GRANT/REVOKE COMMAND; PLEASE
CONSULT THE MANUAL TO SEE WHICH PRIVILEGES CAN BE USED
```

It is important to understand that an event is executed with the privileges of its definer, and that it cannot perform any actions for which its definer does not have the requisite privileges. For example, suppose that `jon@ghidora` has the `EVENT` privilege for `myschema`. Suppose also that this user has the `SELECT` privilege for `myschema`, but no other privileges for this schema. It is possible for `jon@ghidora` to create a new event such as this one:

```
CREATE EVENT e_store_ts
ON SCHEDULE
EVERY 10 SECOND
DO
INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
```

The user waits for a minute or so, and then performs a `SELECT * FROM mytable;` query, expecting to see several new rows in the table. Instead, he finds that the table is empty. Since he does not have the `INSERT` privilege for the table in question, the event has no

effect.

If you inspect the MySQL error log (*hostname.err*), you can see that the event is executing, but the action it is attempting to perform fails, as indicated by `RetCode=0`:

```
060209 22:39:44 [Note] EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:39:44 [Note] EVEX EXECUTED event newdb.e [EXPR:10]. RetCode=0
060209 22:39:54 [Note] EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:39:54 [Note] EVEX EXECUTED event newdb.e [EXPR:10]. RetCode=0
060209 22:40:04 [Note] EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:40:04 [Note] EVEX EXECUTED event newdb.e [EXPR:10]. RetCode=0
```

Since this user very likely does not have access to the error log, he can verify whether the event's action statement is valid by running it himself:

```
mysql> INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
ERROR 1142 (42000): INSERT COMMAND DENIED TO USER
'JON'@'GHIDORA' FOR TABLE 'MYTABLE'
```

Inspection of the `INFORMATION_SCHEMA.EVENTS` table shows that `e_store_ts` exists and is enabled, but its `LAST_EXECUTED` column is `NULL`:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
> WHERE EVENT_NAME='e_store_ts'
> AND EVENT_SCHEMA='myschema'\G
***** 1. row *****
EVENT_CATALOG: NULL
EVENT_SCHEMA: myschema
EVENT_NAME: e_store_ts
DEFINER: jon@ghidora
EVENT_BODY: SQL
EVENT_DEFINITION: INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP())
EVENT_TYPE: RECURRING
EXECUTE_AT: NULL
INTERVAL_VALUE: 5
INTERVAL_FIELD: INTERVAL_SECOND
SQL_MODE: NULL
STARTS: 0000-00-00 00:00:00
ENDS: 0000-00-00 00:00:00
STATUS: ENABLED
ON_COMPLETION: NOT PRESERVE
CREATED: 2006-02-09 22:36:06
LAST_ALTERED: 2006-02-09 22:36:06
LAST_EXECUTED: NULL
EVENT_COMMENT:
1 row in set (0.00 sec)
```

Note

Prior to MySQL 5.1.12, there was no `EVENT_DEFINITION` column, and `EVENT_BODY` contained the SQL statement or statements to be executed. See [Section 23.20, “The INFORMATION_SCHEMA EVENTS Table”](#), for more information.

To rescind the `EVENT` privilege, use the `REVOKE` statement. In this example, the `EVENT` privilege on the schema `myschema` is removed from the `jon@ghidora` user account:

```
REVOKE EVENT ON myschema.* FROM jon@ghidora;
```

Important

Revoking the `EVENT` privilege from a user does not delete or disable any events that may have been created by that user.

An event is not migrated or dropped as a result of the renaming or dropping of the user who created it.

For example, suppose that that user `jon@ghidora` has been granted the `EVENT` and `INSERT` privileges on the `myschema` schema. This user then creates the following event:

```
CREATE EVENT e_insert
ON SCHEDULE
EVERY 7 SECOND
DO
INSERT INTO myschema.mytable;
```

After this event has been created, `root` revokes the `EVENT` privilege for `jon@ghidora`. However, `e_insert` continues to execute,

inserting a new row into `mytable` each seven seconds. The same would be true if `root` had issued either of these statements:

- `DROP USER jon@ghidora;`
- `RENAME USER jon@ghidora TO someotherguy@ghidora;`

You can verify that this is true by examining the `mysql.event` table (discussed later in this section) or the `INFORMATION_SCHEMA.EVENTS` table (see [Section 23.20, “The INFORMATION_SCHEMA EVENTS Table”](#)) before and after issuing a `DROP USER` or `RENAME USER` statement.

Event definitions are stored in the `mysql.event` table, which was added in MySQL 5.1.6. To drop an event created by another user account, the MySQL `root` user (or another user with the necessary privileges) can delete rows from this table. For example, to remove the event `e_insert` shown previously, `root` can use the following statement:

```
DELETE FROM mysql.event
WHERE db = 'myschema'
  AND definer = 'jon@ghidora'
  AND name = 'e_insert';
```

It is very important to match the event name, database schema name, and user account when deleting rows from the `mysql.event` table. This is because the same user can create different events of the same name in different schemas.

Note

The namespace for scheduled events changed in MySQL 5.1.12. Prior to that MySQL version, different users could create different events having the same name in the same database; in MySQL 5.1.12 and later, that is no longer the case. When upgrading to MySQL 5.1.12 or later from MySQL 5.1.11 or earlier, it is extremely important to make sure that no events in the same database share the same name, *prior to performing the upgrade*.

Users' `EVENT` privileges are stored in the `Event_priv` columns of the `mysql.user` and `mysql.db` tables. In both cases, this column holds one of the values 'Y' or 'N'. 'N' is the default. `mysql.user.Event_priv` is set to 'Y' for a given user only if that user has the global `EVENT` privilege (that is, if the privilege was bestowed using `GRANT EVENT ON *.*`). For a schema-level `EVENT` privilege, `GRANT` creates a row in `mysql.db` and sets that row's `Db` column to the name of the schema, the `User` column to the name of the user, and the `Event_priv` column to 'Y'. There should never be any need to manipulate these tables directly, since the `GRANT EVENT` and `REVOKE EVENT` statement perform the required operations on them.

MySQL 5.1.6 introduces five status variables providing counts of event-related operations (but *not* of statements executed by events — see [Section 21.6, “Event Scheduler Limitations and Restrictions”](#)). These are:

- `Com_create_event`: The number of `CREATE EVENT` statements executed since the last server restart.
- `Com_alter_event`: The number of `ALTER EVENT` statements executed since the last server restart.
- `Com_drop_event`: The number of `DROP EVENT` statements executed since the last server restart.
- `Com_show_create_event`: The number of `SHOW CREATE EVENT` statements executed since the last server restart.
- `Com_show_events`: The number of `SHOW EVENTS` statements executed since the last server restart.

You can view current values for all of these at one time by running the statement `SHOW STATUS LIKE '%event%';`.

21.6. Event Scheduler Limitations and Restrictions

This section lists restrictions and limitations applying to event scheduling in MySQL 5.1.

Qualification of identifiers. In MySQL 5.1.6 only, any table referenced in an event's action statement must be fully qualified with the name of the schema in which it occurs (that is, as `schema_name.table_name`).

Case sensitivity of event identifiers. Beginning with MySQL 5.1.8, event names are handled in case-insensitive fashion. For example, this means that you cannot have two events in the same database (and — prior to MySQL 5.1.12 — with the same definer) with the names `anEvent` and `AnEvent`.

Important

If you have events created in MySQL 5.1.7 or earlier which are assigned to the same database and have the same definer, and whose names differ only with respect to lettercase, then you must rename these events to respect case-sensitive handling before upgrading to MySQL 5.1.8 or later.

Modification of events by stored routines and triggers. An event may not be created, altered, or dropped by a trigger, stored routine, or another event. An event also may not create, alter, or drop triggers or stored routines. ([Bug#16409](#), [Bug#18896](#))

Resolution of event timings. Event timings using the intervals [YEAR](#), [QUARTER](#), [MONTH](#), and [YEAR_MONTH](#) are resolved in months; those using any other interval are resolved in seconds. There is no way to cause events scheduled to occur at the same second to execute in a given order. In addition — due to rounding, the nature of threaded applications, and the fact that a non-zero length of time is required to create events and to signal their execution — events may be delayed by as much as 1 or 2 seconds. However, the time shown in the [INFORMATION_SCHEMA.EVENTS](#) table's [LAST_EXECUTED](#) column or the [mysql.event](#) table's [last_executed](#) column is always accurate to within one second of the time the event was actually executed. (See also [Bug#16522](#).)

Effects on statement counts. Each execution of the statements contained in the body of an event takes place in a new connection; thus, these statements has no effect in a given user session on the server's statement counts such as [Com_select](#) and [Com_insert](#) that are displayed by [SHOW STATUS](#). However, such counts are updated in the global scope. ([Bug#16422](#))

Visibility of events belonging to other users. Prior to MySQL 5.1.12, you could not view another user's events in the [INFORMATION_SCHEMA.EVENTS](#) table. In other words, any query made against this table was treated as though it contained the condition [DEFINER = CURRENT_USER\(\)](#) in the [WHERE](#) clause.

Start times. Events cannot be created with a start time that is in the past.

Latest time supported. Events do not support times later than the end of the Unix Epoch; this is approximately the end of the year 2038. Prior to MySQL 5.1.8, handling in scheduled events of dates later than this was buggy; starting with MySQL 5.1.8, such dates are specifically disallowed by the Event Scheduler. ([Bug#16396](#))

Server SQL mode. In MySQL 5.1.6, [INFORMATION_SCHEMA.EVENTS](#) shows [NULL](#) in the [SQL_MODE](#) column. Beginning with MySQL 5.1.7, the [SQL_MODE](#) displayed is that in effect when the event was created.

Dropping or altering events. In MySQL 5.1.6, the only way to drop or alter an event created by a user who was not the definer of that event was by manipulation of the [mysql.event](#) system table by the MySQL [root](#) user or by another user with privileges on this table. Beginning with MySQL 5.1.7, [DROP USER](#) drops all events for which that user was the definer; also beginning with MySQL 5.1.7 [DROP SCHEMA](#) drops all events associated with the dropped schema.

Schema migration not supported. As with stored routines, events are not migrated to the new schema by the [RENAME SCHEMA](#) (or [RENAME DATABASE](#)) statement. See [Section 12.1.18, “RENAME DATABASE Syntax”](#).

Database object references in ON SCHEDULE clauses. References to stored routines, user-defined functions, and tables in the [ON SCHEDULE](#) clauses of [CREATE EVENT](#) and [ALTER EVENT](#) statements are not supported. Beginning with MySQL 5.1.13, these sorts of references are disallowed. (See [Bug#22830](#) for more information.)

Disallowed statements. Generally speaking, statements which are not permitted in stored routines or in SQL prepared statements are also not allowed in the body of an event. See [Section D.1, “Restrictions on Stored Routines and Triggers”](#), and [Section 12.7, “SQL Syntax for Prepared Statements”](#), for more information.

Upgrading to MySQL 5.1.18 or later. When upgrading to MySQL 5.1.18 or later from a previous MySQL version where scheduled events were in use, the upgrade utilities [mysql_upgrade](#) and [mysql_fix_privilege_tables](#) do not accomodate changes in system tables relating to the Event Scheduler. As a workaround, you can dump events before the upgrade, then restore them from the dump afterwards. This issue was fixed in MySQL 5.1.20 (see [Bug#28521](#)).

Chapter 22. Views

Views (including updatable views) are available in MySQL Server 5.1.

Answers to some frequently asked questions concerning views in MySQL 5.1 can be found in [Section A.6, “MySQL 5.1 FAQ — Views”](#).

This chapter discusses the following topics:

- Creating or altering views with [CREATE VIEW](#) or [ALTER VIEW](#)
- Destroying views with [DROP VIEW](#)

Discussion of restrictions on use of views is given in [Section D.4, “Restrictions on Views”](#).

To use views if you have upgraded to MySQL 5.1 from an older release that did not support views, you should upgrade your grant tables so that they contain the view-related privileges. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

Metadata about views can be obtained from the [INFORMATION_SCHEMA.VIEWS](#) table and by using the [SHOW CREATE VIEW](#) statement. See [Section 23.15, “The INFORMATION_SCHEMA VIEWS Table”](#), and [Section 12.5.4.11, “SHOW CREATE VIEW Syntax”](#).

22.1. ALTER VIEW Syntax

```
ALTER
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement changes the definition of a view, which must exist. The syntax is similar to that for [CREATE VIEW](#) and the effect is the same as for [CREATE OR REPLACE VIEW](#). See [Section 22.2, “CREATE VIEW Syntax”](#). This statement requires the [CREATE VIEW](#) and [DROP](#) privileges for the view, and some privilege for each column referred to in the [SELECT](#) statement.

22.2. CREATE VIEW Syntax

```
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement creates a new view, or replaces an existing one if the [OR REPLACE](#) clause is given. If the view does not exist, [CREATE OR REPLACE VIEW](#) is the same as [CREATE VIEW](#). If the view does exist, [CREATE OR REPLACE VIEW](#) is the same as [ALTER VIEW](#). The *select_statement* is a [SELECT](#) statement that provides the definition of the view. The statement can select from base tables or other views.

This statement requires the [CREATE VIEW](#) privilege for the view, and some privilege for each column selected by the [SELECT](#) statement. For columns used elsewhere in the [SELECT](#) statement you must have the [SELECT](#) privilege. If the [OR REPLACE](#) clause is present, you must also have the [DROP](#) privilege for the view.

A view belongs to a database. By default, a new view is created in the default database. To create the view explicitly in a given database, specify the name as *db_name.view_name* when you create it.

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Base tables and views share the same namespace within a database, so a database cannot contain a base table and a view that have the same name.

Views must have unique column names with no duplicates, just like base tables. By default, the names of the columns retrieved by the `SELECT` statement are used for the view column names. To define explicit names for the view columns, the optional `column_list` clause can be given as a list of comma-separated identifiers. The number of names in `column_list` must be the same as the number of columns retrieved by the `SELECT` statement.

Columns retrieved by the `SELECT` statement can be simple references to table columns. They can also be expressions that use functions, constant values, operators, and so forth.

Unqualified table or view names in the `SELECT` statement are interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the proper database name.

A view can be created from many kinds of `SELECT` statements. It can refer to base tables or other views. It can use joins, `UNION`, and subqueries. The `SELECT` need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+-----+-----+-----+
| qty | price | value |
+-----+-----+-----+
| 3   | 50    | 150   |
+-----+-----+-----+
```

A view definition is subject to the following restrictions:

- The `SELECT` statement cannot contain a subquery in the `FROM` clause.
- The `SELECT` statement cannot refer to system or user variables.
- The `SELECT` statement cannot refer to prepared statement parameters.
- Within a stored routine, the definition cannot refer to routine parameters or local variables.
- Any table or view referred to in the definition must exist. However, after a view has been created, it is possible to drop a table or view that the definition refers to. In this case, use of the view results in an error. To check a view definition for problems of this kind, use the `CHECK TABLE` statement.
- The definition cannot refer to a `TEMPORARY` table, and you cannot create a `TEMPORARY` view.
- The tables named in the view definition must already exist.
- You cannot associate a trigger with a view.

`ORDER BY` is allowed in a view definition, but it is ignored if you select from a view using a statement that has its own `ORDER BY`.

For other options or clauses in the definition, they are added to the options or clauses of the statement that references the view, but the effect is undefined. For example, if a view definition includes a `LIMIT` clause, and you select from the view using a statement that has its own `LIMIT` clause, it is undefined which limit applies. This same principle applies to options such as `ALL`, `DISTINCT`, or `SQL_SMALL_RESULT` that follow the `SELECT` keyword, and to clauses such as `INTO`, `FOR UPDATE`, `LOCK IN SHARE MODE`, and `PROCEDURE`.

If you create a view and then change the query processing environment by changing system variables, that may affect the results that you get from the view:

```
mysql> CREATE VIEW v (mycol) AS SELECT 'abc';
Query OK, 0 rows affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-----+
| mycol |
+-----+
| mycol |
+-----+
1 row in set (0.01 sec)
```

```
mysql> SET sql_mode = 'ANSI_QUOTES';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-----+
| mycol |
+-----+
| abc   |
+-----+
1 row in set (0.00 sec)
```

The **DEFINER** and **SQL SECURITY** clauses specify the security context to be used when checking access privileges at view invocation time. They were added in MySQL 5.1.2.

CURRENT_USER also can be given as **CURRENT_USER()**.

Within a stored routine that is defined with the **SQL SECURITY DEFINER** characteristic, **CURRENT_USER** returns the routine creator. This also affects a view defined within such a routine, if the view definition contains a **DEFINER** value of **CURRENT_USER**.

The default **DEFINER** value is the user who executes the **CREATE VIEW** statement. (This is the same as **DEFINER = CURRENT_USER**.) If a *user* value is given, it should be a MySQL account in '*user_name*'@'*host_name*' format (the same format used in the **GRANT** statement). The *user_name* and *host_name* values both are required.

If you specify the **DEFINER** clause, you cannot set the value to any user but your own unless you have the **SUPER** privilege. These rules determine the legal **DEFINER** user values:

- If you do not have the **SUPER** privilege, the only legal *user* value is your own account, either specified literally or by using **CURRENT_USER**. You cannot set the definer to some other account.
- If you have the **SUPER** privilege, you can specify any syntactically legal account name. If the account does not actually exist, a warning is generated.

The **SQL SECURITY** characteristic determines which MySQL account to use when checking access privileges for the view when the view is executed. The legal characteristic values are **DEFINER** and **INVOKER**. These indicate that the view must be executable by the user who defined it or invoked it, respectively. The default **SQL SECURITY** value is **DEFINER**.

As of MySQL 5.1.2 (when the **DEFINER** and **SQL SECURITY** clauses were implemented), view privileges are checked like this:

- At view definition time, the view creator must have the privileges needed to use the top-level objects accessed by the view. For example, if the view definition refers to a stored function, only the privileges needed to invoke the function can be checked. The privileges required when the function runs can be checked only as it executes: For different invocations of the function, different execution paths within the function might be taken.
- At view execution time, privileges for objects accessed by the view are checked against the privileges held by the view creator or invoker, depending on whether the **SQL SECURITY** characteristic is **DEFINER** or **INVOKER**, respectively.
- If view execution causes execution of a stored function, privilege checking for statements executed within the function depend on whether the function is defined with a **SQL SECURITY** characteristic of **DEFINER** or **INVOKER**. If the security characteristic is **DEFINER**, the function runs with the privileges of its creator. If the characteristic is **INVOKER**, the function runs with the privileges determined by the view's **SQL SECURITY** characteristic.

Prior to MySQL 5.1.2 (before the **DEFINER** and **SQL SECURITY** clauses were implemented), privileges required for objects used in a view are checked at view creation time.

Example: A view might depend on a stored function, and that function might invoke other stored routines. For example, the following view invokes a stored function **f()**:

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);
```

Suppose that **f()** contains a statement such as this:

```
IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```


The privileges required for executing statements within `f()` need to be checked when `f()` executes. This might mean that privileges are needed for `p1()` or `p2()`, depending on the execution path within `f()`. Those privileges need to be checked at runtime, and the user who must possess the privileges is determined by the `SQL SECURITY` values of the function `f()` and the view `v`.

The `DEFINER` and `SQL SECURITY` clauses for views are extensions to standard SQL. In standard SQL, views are handled using the rules for `SQL SECURITY INVOKER`.

If you invoke a view that was created before MySQL 5.0.13/5.1.2, it is treated as though it was created with a `SQL SECURITY DEFINER` clause and with a `DEFINER` value that is the same as your account. However, because the actual definer is unknown, MySQL issues a warning. To make the warning go away, it is sufficient to re-create the view so that the view definition includes a `DEFINER` clause.

The optional `ALGORITHM` clause is a MySQL extension to standard SQL. `ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`. The default algorithm is `UNDEFINED` if no `ALGORITHM` clause is present. The algorithm affects how MySQL processes the view.

For `MERGE`, the text of a statement that refers to the view and the view definition are merged such that parts of the view definition replace corresponding parts of the statement.

For `TEMPTABLE`, the results from the view are retrieved into a temporary table, which then is used to execute the statement.

For `UNDEFINED`, MySQL chooses which algorithm to use. It prefers `MERGE` over `TEMPTABLE` if possible, because `MERGE` is usually more efficient and because a view cannot be updatable if a temporary table is used.

A reason to choose `TEMPTABLE` explicitly is that locks can be released on underlying tables after the temporary table has been created and before it is used to finish processing the statement. This might result in quicker lock release than the `MERGE` algorithm so that other clients that use the view are not blocked as long.

A view algorithm can be `UNDEFINED` for three reasons:

- No `ALGORITHM` clause is present in the `CREATE VIEW` statement.
- The `CREATE VIEW` statement has an explicit `ALGORITHM = UNDEFINED` clause.
- `ALGORITHM = MERGE` is specified for a view that can be processed only with a temporary table. In this case, MySQL generates a warning and sets the algorithm to `UNDEFINED`.

As mentioned earlier, `MERGE` is handled by merging corresponding parts of a view definition into the statement that refers to the view. The following examples briefly illustrate how the `MERGE` algorithm works. The examples assume that there is a view `v_merge` that has this definition:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 1: Suppose that we issue this statement:

```
SELECT * FROM v_merge;
```

MySQL handles the statement as follows:

- `v_merge` becomes `t`
- `*` becomes `vc1, vc2`, which corresponds to `c1, c2`
- The view `WHERE` clause is added

The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 2: Suppose that we issue this statement:

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

This statement is handled similarly to the previous one, except that `vc1 < 100` becomes `c1 < 100` and the view `WHERE` clause is added to the statement `WHERE` clause using an `AND` connective (and parentheses are added to make sure the parts of the clause are executed with correct precedence). The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

Effectively, the statement to be executed has a `WHERE` clause of this form:

```
WHERE (select WHERE) AND (view WHERE)
```

The `MERGE` algorithm requires a one-to-one relationship between the rows in the view and the rows in the underlying table. If this relationship does not hold, a temporary table must be used instead. Lack of a one-to-one relationship occurs if the view contains any of a number of constructs:

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Refers only to literal values (in this case, there is no underlying table)

Some views are updatable. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view non-updatable. To be more specific, a view is not updatable if it contains any of the following:

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Certain joins (see additional join discussion later in this section)
- Non-updatable view in the `FROM` clause
- A subquery in the `WHERE` clause that refers to a table in the `FROM` clause
- Refers only to literal values (in this case, there is no underlying table to update)
- `ALGORITHM = TEMPTABLE` (use of a temporary table always makes a view non-updatable)

With respect to insertability (being updatable with `INSERT` statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

- There must be no duplicate view column names.
- The view must contain all columns in the base table that do not have a default value.
- The view columns must be simple column references and not derived columns. A derived column is one that is not a simple column reference but is derived from an expression. These are examples of derived columns:

```
3.14159
col1 + 3
UPPER(col2)
col3 / col4
(subquery)
```

A view that has a mix of simple column references and derived columns is not insertable, but it can be updatable if you update only those columns that are not derived. Consider this view:

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

This view is not insertable because `col2` is derived from an expression. But it is updatable if the update does not try to update `col2`. This update is allowable:

```
UPDATE v SET col1 = 0;
```

This update is not allowable because it attempts to update a derived column:

```
UPDATE v SET col2 = 0;
```

It is sometimes possible for a multiple-table view to be updatable, assuming that it can be processed with the `MERGE` algorithm. For this to work, the view must use an inner join (not an outer join or a `UNION`). Also, only a single table in the view definition can be updated, so the `SET` clause must name only columns from one of the tables in the view. Views that use `UNION ALL` are disallowed even though they might be theoretically updatable, because the implementation uses temporary tables to process them.

For a multiple-table updatable view, `INSERT` can work if it inserts into a single table. `DELETE` is not supported.

`INSERT DELAYED` is not supported for views.

If a table contains an `AUTO_INCREMENT` column, inserting into an insertable view on the table that does not include the `AUTO_INCREMENT` column does not change the value of `LAST_INSERT_ID()`, because the side effects of inserting default values into columns not part of the view should not be visible.

The `WITH CHECK OPTION` clause can be given for an updatable view to prevent inserts or updates to rows except those for which the `WHERE` clause in the *select_statement* is true.

In a `WITH CHECK OPTION` clause for an updatable view, the `LOCAL` and `CASCADED` keywords determine the scope of check testing when the view is defined in terms of another view. The `LOCAL` keyword restricts the `CHECK OPTION` only to the view being defined. `CASCADED` causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is `CASCADED`. Consider the definitions for the following table and set of views:

```
mysql> CREATE TABLE t1 (a INT);
mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
-> WITH CHECK OPTION;
mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
-> WITH LOCAL CHECK OPTION;
mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
-> WITH CASCADED CHECK OPTION;
```

Here the `v2` and `v3` views are defined in terms of another view, `v1`. `v2` has a `LOCAL` check option, so inserts are tested only against the `v2` check. `v3` has a `CASCADED` check option, so inserts are tested not only against its own check, but against those of underlying views. The following statements illustrate these differences:

```
mysql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE`

(and similar operations) are legal for the view. Otherwise, the flag is set to **NO** (false). The **IS_UPDATABLE** column in the **INFORMATION_SCHEMA.VIEWS** table displays the status of this flag. It means that the server always knows whether a view is updatable. If the view is not updatable, statements such **UPDATE**, **DELETE**, and **INSERT** are illegal and will be rejected. (Note that even if a view is updatable, it might not be possible to insert into it, as described elsewhere in this section.)

The updatability of views may be affected by the value of the **updatable_views_with_limit** system variable. See [Section 5.2.3, “System Variables”](#).

22.3. DROP VIEW Syntax

```
DROP VIEW [IF EXISTS]
  view_name [, view_name] ...
[RESTRICT | CASCADE]
```

DROP VIEW removes one or more views. You must have the **DROP** privilege for each view. If any of the views named in the argument list do not exist, MySQL returns an error indicating by name which non-existing views it was unable to drop, but it also drops all of the views in the list that do exist.

The **IF EXISTS** clause prevents an error from occurring for views that don't exist. When this clause is given, a **NOTE** is generated for each non-existent view. See [Section 12.5.4.32, “SHOW WARNINGS Syntax”](#).

RESTRICT and **CASCADE**, if given, are parsed and ignored.

Chapter 23. The `INFORMATION_SCHEMA` Database

`INFORMATION_SCHEMA` provides access to database metadata.

Metadata is data about the data, such as the name of a database or table, the data type of a column, or access privileges. Other terms that sometimes are used for this information are *data dictionary* and *system catalog*.

`INFORMATION_SCHEMA` is the information database, the place that stores information about all the other databases that the MySQL server maintains. Inside `INFORMATION_SCHEMA` there are several read-only tables. They are actually views, not base tables, so there are no files associated with them.

In effect, we have a database named `INFORMATION_SCHEMA`, although the server does not create a database directory with that name. It is possible to select `INFORMATION_SCHEMA` as the default database with a `USE` statement, but it is possible only to read the contents of tables. You cannot insert into them, update them, or delete from them.

Here is an example of a statement that retrieves information from `INFORMATION_SCHEMA`:

```
mysql> SELECT table_name, table_type, engine
-> FROM information_schema.tables
-> WHERE table_schema = 'db5'
-> ORDER BY table_name DESC;
```

table_name	table_type	engine
v56	VIEW	NULL
v3	VIEW	NULL
v2	VIEW	NULL
v	VIEW	NULL
tables	BASE TABLE	MyISAM
t7	BASE TABLE	MyISAM
t3	BASE TABLE	MyISAM
t2	BASE TABLE	MyISAM
t	BASE TABLE	MyISAM
pk	BASE TABLE	InnoDB
loop	BASE TABLE	MyISAM
kurs	BASE TABLE	MyISAM
k	BASE TABLE	MyISAM
into	BASE TABLE	MyISAM
goto	BASE TABLE	MyISAM
fk2	BASE TABLE	InnoDB
fk	BASE TABLE	InnoDB

17 rows in set (0.01 sec)

Explanation: The statement requests a list of all the tables in database `db5`, in reverse alphabetical order, showing just three pieces of information: the name of the table, its type, and its storage engine.

Each MySQL user has the right to access these tables, but can see only the rows in the tables that correspond to objects for which the user has the proper access privileges. In some cases (for example, the `ROUTINE_DEFINITION` column in the `INFORMATION_SCHEMA.ROUTINES` table), users who have insufficient privileges will see `NULL`.

The `SELECT ... FROM INFORMATION_SCHEMA` statement is intended as a more consistent way to provide access to the information provided by the various `SHOW` statements that MySQL supports (`SHOW DATABASES`, `SHOW TABLES`, and so forth). Using `SELECT` has these advantages, compared to `SHOW`:

- It conforms to Codd's rules. That is, all access is done on tables.
- Nobody needs to learn a new statement syntax. Because they already know how `SELECT` works, they only need to learn the object names.
- The implementor need not worry about adding keywords.
- There are millions of possible output variations, instead of just one. This provides more flexibility for applications that have varying requirements about what metadata they need.
- Migration is easier because every other DBMS does it this way.

However, because `SHOW` is popular with MySQL employees and users, and because it might be confusing were it to disappear, the advantages of conventional syntax are not a sufficient reason to eliminate `SHOW`. In fact, along with the implementation of `INFORMA-`

`INFORMATION_SCHEMA`, there are enhancements to `SHOW` as well. These are described in [Section 23.27, “Extensions to SHOW Statements”](#).

There is no difference between the privileges required for `SHOW` statements and those required to select information from `INFORMATION_SCHEMA`. In either case, you have to have some privilege on an object in order to see information about it.

The implementation for the `INFORMATION_SCHEMA` table structures in MySQL follows the ANSI/ISO SQL:2003 standard Part 11 *Schemata*. Our intent is approximate compliance with SQL:2003 core feature F021 *Basic information schema*.

Users of SQL Server 2000 (which also follows the standard) may notice a strong similarity. However, MySQL has omitted many columns that are not relevant for our implementation, and added columns that are MySQL-specific. One such column is the `ENGINE` column in the `INFORMATION_SCHEMA.TABLES` table.

Although other DBMSs use a variety of names, like `syscat` or `system`, the standard name is `INFORMATION_SCHEMA`.

The following sections describe each of the tables and columns that are in `INFORMATION_SCHEMA`. For each column, there are three pieces of information:

- “`INFORMATION_SCHEMA` Name” indicates the name for the column in the `INFORMATION_SCHEMA` table. This corresponds to the standard SQL name unless the “Remarks” field says “MySQL extension.”
- “`SHOW` Name” indicates the equivalent field name in the closest `SHOW` statement, if there is one.
- “Remarks” provides additional information where applicable. If this field is `NULL`, it means that the value of the column is always `NULL`. If this field says “MySQL extension,” the column is a MySQL extension to standard SQL.

To avoid using any name that is reserved in the standard or in DB2, SQL Server, or Oracle, we changed the names of some columns marked “MySQL extension”. (For example, we changed `COLLATION` to `TABLE_COLLATION` in the `TABLES` table.) See the list of reserved words near the end of this article: <http://web.archive.org/web/20030201202307/www.dbazine.com/gulutzan5.html>.

The definition for character columns (for example, `TABLES.TABLE_NAME`) is generally `VARCHAR(N) CHARACTER SET utf8` where `N` is at least 64. MySQL uses the default collation for this character set (`utf8_general_ci`) for all searches, sorts, comparisons, and other string operations on such columns. If the default collation is not correct for your needs, you can force a suitable collation with a `COLLATE` clause ([Section 9.5.1, “Using COLLATE in SQL Statements”](#)).

Each section indicates what `SHOW` statement is equivalent to a `SELECT` that retrieves information from `INFORMATION_SCHEMA`, if there is such a statement.

Note

At present, there are some missing columns and some columns out of order. We are working on this and updating the documentation as changes are made.

For answers to questions that are often asked concerning the `INFORMATION_SCHEMA` database, see [Section A.7, “MySQL 5.0 FAQ — INFORMATION_SCHEMA”](#).

23.1. The `INFORMATION_SCHEMA` `SCHEMATA` Table

A schema is a database, so the `SCHEMATA` table provides information about databases.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>CATALOG_NAME</code>		<code>NULL</code>
<code>SCHEMA_NAME</code>		Database
<code>DEFAULT_CHARACTER_SET_NAME</code>		
<code>DEFAULT_COLLATION_NAME</code>		
<code>SQL_PATH</code>		<code>NULL</code>

The following statements are equivalent:

```
SELECT SCHEMA_NAME AS `Database`
FROM INFORMATION_SCHEMA.SCHEMATA
[WHERE SCHEMA_NAME LIKE 'wild']
```

```
SHOW DATABASES
[LIKE 'wild']
```

23.2. The `INFORMATION_SCHEMA TABLES` Table

The `TABLES` table provides information about tables in databases.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>TABLE_CATALOG</code>		<code>NULL</code>
<code>TABLE_SCHEMA</code>	<code>Table_...</code>	
<code>TABLE_NAME</code>	<code>Table_...</code>	
<code>TABLE_TYPE</code>		
<code>ENGINE</code>	<code>Engine</code>	MySQL extension
<code>VERSION</code>	<code>Version</code>	MySQL extension
<code>ROW_FORMAT</code>	<code>Row_format</code>	MySQL extension
<code>TABLE_ROWS</code>	<code>Rows</code>	MySQL extension
<code>AVG_ROW_LENGTH</code>	<code>Avg_row_length</code>	MySQL extension
<code>DATA_LENGTH</code>	<code>Data_length</code>	MySQL extension
<code>MAX_DATA_LENGTH</code>	<code>Max_data_length</code>	MySQL extension
<code>INDEX_LENGTH</code>	<code>Index_length</code>	MySQL extension
<code>DATA_FREE</code>	<code>Data_free</code>	MySQL extension
<code>AUTO_INCREMENT</code>	<code>Auto_increment</code>	MySQL extension
<code>CREATE_TIME</code>	<code>Create_time</code>	MySQL extension
<code>UPDATE_TIME</code>	<code>Update_time</code>	MySQL extension
<code>CHECK_TIME</code>	<code>Check_time</code>	MySQL extension
<code>TABLE_COLLATION</code>	<code>Collation</code>	MySQL extension
<code>CHECKSUM</code>	<code>Checksum</code>	MySQL extension
<code>CREATE_OPTIONS</code>	<code>Create_options</code>	MySQL extension
<code>TABLE_COMMENT</code>	<code>Comment</code>	MySQL extension

Notes:

- `TABLE_SCHEMA` and `TABLE_NAME` are a single field in a `SHOW` display, for example `Table_in_db1`.
- `TABLE_TYPE` should be `BASE TABLE` or `VIEW`. If table is temporary, then `TABLE_TYPE` = `TEMPORARY`. (There are no temporary views, so this is not ambiguous.)
- For partitioned tables, beginning with MySQL 5.1.9, the `ENGINE` column shows the name of the storage engine used by all partitions. (Previously, this column showed `PARTITION` for such tables.)
- The `TABLE_ROWS` column is `NULL` if the table is in the `INFORMATION_SCHEMA` database. For `InnoDB` tables, the row count is only a rough estimate used in SQL optimization.
- For tables using the `NDBCLUSTER` storage engine, beginning with MySQL 5.1.12, the `DATA_LENGTH` column reflects the true amount of storage for variable-width columns. (See [Bug#18413](#).)

Note

Because MySQL Cluster allocates storage for variable-width columns in 10-page extents of 32 kilobytes each, space usage for such columns is reported in increments of 320 KB.

- We have nothing for the table's default character set. `TABLE_COLLATION` is close, because collation names begin with a character set name.

- Beginning with MySQL 5.1.9, the `CREATE_OPTIONS` column shows `partitioned` if the table is partitioned.

The following statements are equivalent:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
[WHERE table_schema = 'db_name']
[WHERE|AND table_name LIKE 'wild']

SHOW TABLES
[FROM db_name]
[LIKE 'wild']
```

23.3. The INFORMATION_SCHEMA COLUMNS Table

The `COLUMNS` table provides information about columns in tables.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		see notes
COLUMN_DEFAULT	Default	
IS_NULLABLE	Null	
DATA_TYPE	Type	
CHARACTER_MAXIMUM_LENGTH	Type	
CHARACTER_OCTET_LENGTH		
NUMERIC_PRECISION	Type	
NUMERIC_SCALE	Type	
CHARACTER_SET_NAME		
COLLATION_NAME	Collation	
COLUMN_TYPE	Type	MySQL extension
COLUMN_KEY	Key	MySQL extension
EXTRA	Extra	MySQL extension
COLUMN_COMMENT	Comment	MySQL extension

Notes:

- In `SHOW`, the `Type` display includes values from several different `COLUMNS` columns.
- `ORDINAL_POSITION` is necessary because you might want to say `ORDER BY ORDINAL_POSITION`. Unlike `SHOW`, `SELECT` does not have automatic ordering.
- `CHARACTER_OCTET_LENGTH` should be the same as `CHARACTER_MAXIMUM_LENGTH`, except for multi-byte character sets.
- `CHARACTER_SET_NAME` can be derived from `Collation`. For example, if you say `SHOW FULL COLUMNS FROM t`, and you see in the `Collation` column a value of `latin1_swedish_ci`, the character set is what's before the first underscore: `latin1`.

The following statements are nearly equivalent:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'tbl_name'
[AND table_schema = 'db_name']
[AND column_name LIKE 'wild']
```



```
SHOW COLUMNS
FROM tbl_name
[FROM db_name]
[LIKE 'wild']
```

23.4. The INFORMATION_SCHEMA STATISTICS Table

The `STATISTICS` table provides information about table indexes.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		= Database
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		= Database
INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	MySQL extension
PACKED	Packed	MySQL extension
NULLABLE	Null	MySQL extension
INDEX_TYPE	Index_type	MySQL extension
COMMENT	Comment	MySQL extension

Notes:

- There is no standard table for indexes. The preceding list is similar to what SQL Server 2000 returns for `sp_statistics`, except that we replaced the name `QUALIFIER` with `CATALOG` and we replaced the name `OWNER` with `SCHEMA`.

Clearly, the preceding table and the output from `SHOW INDEX` are derived from the same parent. So the correlation is already close.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
WHERE table_name = 'tbl_name'
[AND table_schema = 'db_name']

SHOW INDEX
FROM tbl_name
[FROM db_name]
```

23.5. The INFORMATION_SCHEMA USER_PRIVILEGES Table

The `USER_PRIVILEGES` table provides information about global privileges. This information comes from the `mysql.user` grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		' <i>user_name</i> '@' <i>host_name</i> ' value, MySQL extension
TABLE_CATALOG		NULL, MySQL extension
PRIVILEGE_TYPE		MySQL extension

<code>IS_GRANTABLE</code>		MySQL extension

Notes:

- This is a non-standard table. It takes its values from the `mysql.user` table.

23.6. The `INFORMATION_SCHEMA SCHEMA_PRIVILEGES` Table

The `SCHEMA_PRIVILEGES` table provides information about schema (database) privileges. This information comes from the `mysql.db` grant table.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>GRANTEE</code>		' <i>user_name</i> '@' <i>host_name</i> ' value, MySQL extension
<code>TABLE_CATALOG</code>		<code>NULL</code> , MySQL extension
<code>TABLE_SCHEMA</code>		MySQL extension
<code>PRIVILEGE_TYPE</code>		MySQL extension
<code>IS_GRANTABLE</code>		MySQL extension

Notes:

- This is a non-standard table. It takes its values from the `mysql.db` table.

23.7. The `INFORMATION_SCHEMA TABLE_PRIVILEGES` Table

The `TABLE_PRIVILEGES` table provides information about table privileges. This information comes from the `mysql.tables_priv` grant table.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>GRANTEE</code>		' <i>user_name</i> '@' <i>host_name</i> ' value
<code>TABLE_CATALOG</code>		<code>NULL</code>
<code>TABLE_SCHEMA</code>		
<code>TABLE_NAME</code>		
<code>PRIVILEGE_TYPE</code>		
<code>IS_GRANTABLE</code>		

Notes:

- `PRIVILEGE_TYPE` can contain one (and only one) of these values: `SELECT`, `INSERT`, `UPDATE`, `REFERENCES`, `ALTER`, `INDEX`, `DROP`, `CREATE VIEW`.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
SHOW GRANTS ...
```

23.8. The `INFORMATION_SCHEMA COLUMN_PRIVILEGES` Table

The `COLUMN_PRIVILEGES` table provides information about column privileges. This information comes from the `mysql.columns_priv` grant table.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>GRANTEE</code>		' <i>user_name</i> '@' <i>host_name</i> ' value
<code>TABLE_CATALOG</code>		NULL
<code>TABLE_SCHEMA</code>		
<code>TABLE_NAME</code>		
<code>COLUMN_NAME</code>		
<code>PRIVILEGE_TYPE</code>		
<code>IS_GRANTABLE</code>		

Notes:

- In the output from `SHOW FULL COLUMNS`, the privileges are all in one field and in lowercase, for example, `select, insert, update, references`. In `COLUMN_PRIVILEGES`, there is one privilege per row, in uppercase.
- `PRIVILEGE_TYPE` can contain one (and only one) of these values: `SELECT`, `INSERT`, `UPDATE`, `REFERENCES`.
- If the user has `GRANT OPTION` privilege, `IS_GRANTABLE` should be `YES`. Otherwise, `IS_GRANTABLE` should be `NO`. The output does not list `GRANT OPTION` as a separate privilege.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
SHOW GRANTS ...
```

23.9. The `INFORMATION_SCHEMA CHARACTER_SETS` Table

The `CHARACTER_SETS` table provides information about available character sets.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>CHARACTER_SET_NAME</code>	Charset	
<code>DEFAULT_COLLATE_NAME</code>	Default collation	
<code>DESCRIPTION</code>	Description	MySQL extension
<code>MAXLEN</code>	Maxlen	MySQL extension

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
[WHERE name LIKE 'wild']
SHOW CHARACTER SET
[LIKE 'wild']
```

23.10. The `INFORMATION_SCHEMA COLLATIONS` Table

The `COLLATIONS` table provides information about collations for each character set.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>COLLATION_NAME</code>	Collation	
<code>CHARACTER_SET_NAME</code>	Charset	MySQL extension

ID	Id	MySQL extension
IS_DEFAULT	Default	MySQL extension
IS_COMPILED	Compiled	MySQL extension
SORTLEN	Sortlen	MySQL extension

The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
[WHERE collation_name LIKE 'wild']

SHOW COLLATION
[LIKE 'wild']
```

23.11. The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table

The `COLLATION_CHARACTER_SET_APPLICABILITY` table indicates what character set is applicable for what collation. The columns are equivalent to the first two display fields that we get from `SHOW COLLATION`.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	

23.12. The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table

The `TABLE_CONSTRAINTS` table describes which tables have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_SCHEMA		
TABLE_NAME		
CONSTRAINT_TYPE		

Notes:

- The `CONSTRAINT_TYPE` value can be `UNIQUE`, `PRIMARY KEY`, or `FOREIGN KEY`.
- The `UNIQUE` and `PRIMARY KEY` information is about the same as what you get from the `key_name` field in the output from `SHOW INDEX` when the `Non_unique` field is 0.
- The `CONSTRAINT_TYPE` column can contain one of these values: `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`, `CHECK`. This is a `CHAR` (not `ENUM`) column. The `CHECK` value is not available until we support `CHECK`.

23.13. The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table

The `KEY_COLUMN_USAGE` table describes which key columns have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		NULL

CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_CATALOG		
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
ORDINAL_POSITION		
POSITION_IN_UNIQUE_CONSTRAINT		
REFERENCED_TABLE_SCHEMA		
REFERENCED_TABLE_NAME		
REFERENCED_COLUMN_NAME		

Notes:

- If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.
- The value of **ORDINAL_POSITION** is the column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.
- The value of **POSITION_IN_UNIQUE_CONSTRAINT** is **NULL** for unique and primary-key constraints. For foreign-key constraints, it is the ordinal position in key of the table that is being referenced.

For example, suppose that there are two tables name **t1** and **t3** that have the following definitions:

```
CREATE TABLE t1
(
  s1 INT,
  s2 INT,
  s3 INT,
  PRIMARY KEY(s3)
) ENGINE=InnoDB;

CREATE TABLE t3
(
  s1 INT,
  s2 INT,
  s3 INT,
  KEY(s1),
  CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
) ENGINE=InnoDB;
```

For those two tables, the **KEY_COLUMN_USAGE** table has two rows:

- One row with **CONSTRAINT_NAME** = 'PRIMARY', **TABLE_NAME** = 't1', **COLUMN_NAME** = 's3', **ORDINAL_POSITION** = 1, **POSITION_IN_UNIQUE_CONSTRAINT** = **NULL**.
- One row with **CONSTRAINT_NAME** = 'CO', **TABLE_NAME** = 't3', **COLUMN_NAME** = 's2', **ORDINAL_POSITION** = 1, **POSITION_IN_UNIQUE_CONSTRAINT** = 1.

23.14. The INFORMATION_SCHEMA ROUTINES Table

The **ROUTINES** table provides information about stored routines (both procedures and functions). The **ROUTINES** table does not include user-defined functions (UDFs) at this time.

The column named “**mysql.proc name**” indicates the **mysql.proc** table column that corresponds to the **INFORMATION_SCHEMA.ROUTINES** table column, if any.

INFORMATION_SCHEMA Name	mysql.proc Name	Remarks
SPECIFIC_NAME	specific_name	
ROUTINE_CATALOG		NULL

ROUTINE_SCHEMA	db	
ROUTINE_NAME	name	
ROUTINE_TYPE	type	{PROCEDURE FUNCTION}
DTD_IDENTIFIER		(data type descriptor)
ROUTINE_BODY		SQL
ROUTINE_DEFINITION	body	
EXTERNAL_NAME		NULL
EXTERNAL_LANGUAGE	language	NULL
PARAMETER_STYLE		SQL
IS_DETERMINISTIC	is_deterministic	
SQL_DATA_ACCESS	sql_data_access	
SQL_PATH		NULL
SECURITY_TYPE	security_type	
CREATED	created	
LAST_ALTERED	modified	
SQL_MODE	sql_mode	MySQL extension
ROUTINE_COMMENT	comment	MySQL extension
DEFINER	definer	MySQL extension
CHARACTER_SET_CLIENT		MySQL extension
COLLATION_CONNECTION		MySQL extension
DATABASE_COLLATION		MySQL extension

Notes:

- MySQL calculates `EXTERNAL_LANGUAGE` thus:
 - If `mysql.proc.language='SQL'`, `EXTERNAL_LANGUAGE` is `NULL`
 - Otherwise, `EXTERNAL_LANGUAGE` is what is in `mysql.proc.language`. However, we do not have external languages yet, so it is always `NULL`.
- `CHARACTER_SET_CLIENT` is the session value of the `character_set_client` system variable when the routine was created. `COLLATION_CONNECTION` is the session value of the `collation_connection` system variable when the routine was created. `DATABASE_COLLATION` is the collation of the database with which the routine is associated. These columns were added in MySQL 5.1.21.

23.15. The INFORMATION_SCHEMA VIEWS Table

The `VIEWS` table provides information about views in databases. You must have the `SHOW VIEW` privilege to access this table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
VIEW_DEFINITION		
CHECK_OPTION		
IS_UPDATABLE		
DEFINER		
SECURITY_TYPE		

<code>CHARACTER_SET_CLIENT</code>		MySQL extension
<code>COLLATION_CONNECTION</code>		MySQL extension

Notes:

- The `VIEW_DEFINITION` column has most of what you see in the `Create Table` field that `SHOW CREATE VIEW` produces. Skip the words before `SELECT` and skip the words `WITH CHECK OPTION`. Suppose that the original statement was:

```
CREATE VIEW v AS
  SELECT s2,s1 FROM t
 WHERE s1 > 5
 ORDER BY s1
 WITH CHECK OPTION;
```

Then the view definition looks like this:

```
SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
```

- The `CHECK_OPTION` column always has a value of `NONE`.
- MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE` (and similar operations) are legal for the view. Otherwise, the flag is set to `NO` (false). The `IS_UPDATABLE` column in the `VIEWS` table displays the status of this flag. It means that the server always knows whether a view is updatable. If the view is not updatable, statements such `UPDATE`, `DELETE`, and `INSERT` are illegal and will be rejected. (Note that even if a view is updatable, it might not be possible to insert into it; for details, refer to [Section 22.2, “CREATE VIEW Syntax”](#).)
- The `DEFINER` column indicates who defined the view. `SECURITY_TYPE` has a value of `DEFINER` or `INVOKER`.
- `CHARACTER_SET_CLIENT` is the session value of the `character_set_client` system variable when the view was created. `COLLATION_CONNECTION` is the session value of the `collation_connection` system variable when the view was created. These columns were added in MySQL 5.1.21.

23.16. The `INFORMATION_SCHEMA TRIGGERS` Table

The `TRIGGERS` table provides information about triggers. You must have the `SUPER` privilege to access this table.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>TRIGGER_CATALOG</code>		<code>NULL</code>
<code>TRIGGER_SCHEMA</code>		
<code>TRIGGER_NAME</code>	Trigger	
<code>EVENT_MANIPULATION</code>	Event	
<code>EVENT_OBJECT_CATALOG</code>		<code>NULL</code>
<code>EVENT_OBJECT_SCHEMA</code>		
<code>EVENT_OBJECT_TABLE</code>	Table	
<code>ACTION_ORDER</code>		0
<code>ACTION_CONDITION</code>		<code>NULL</code>
<code>ACTION_STATEMENT</code>	Statement	
<code>ACTION_ORIENTATION</code>		<code>ROW</code>
<code>ACTION_TIMING</code>	Timing	
<code>ACTION_REFERENCE_OLD_TABLE</code>		<code>NULL</code>
<code>ACTION_REFERENCE_NEW_TABLE</code>		<code>NULL</code>
<code>ACTION_REFERENCE_OLD_ROW</code>		<code>OLD</code>
<code>ACTION_REFERENCE_NEW_ROW</code>		<code>NEW</code>

CREATED		NULL (0)
SQL_MODE		MySQL extension
DEFINER		MySQL extension
CHARACTER_SET_CLIENT		MySQL extension
COLLATION_CONNECTION		MySQL extension
DATABASE_COLLATION		MySQL extension

Notes:

- The `TRIGGER_SCHEMA` and `TRIGGER_NAME` columns contain the name of the database in which the trigger occurs and the trigger name, respectively.
- The `EVENT_MANIPULATION` column contains one of the values 'INSERT', 'DELETE', or 'UPDATE'.
- As noted in [Chapter 20, Triggers](#), every trigger is associated with exactly one table. The `EVENT_OBJECT_SCHEMA` and `EVENT_OBJECT_TABLE` columns contain the database in which this table occurs, and the table's name.
- The `ACTION_ORDER` statement contains the ordinal position of the trigger's action within the list of all similar triggers on the same table. Currently, this value is always 0, because it is not possible to have more than one trigger with the same `EVENT_MANIPULATION` and `ACTION_TIMING` on the same table.
- The `ACTION_STATEMENT` column contains the statement to be executed when the trigger is invoked. This is the same as the text displayed in the `Statement` column of the output from `SHOW TRIGGERS`. Note that this text uses UTF-8 encoding.
- The `ACTION_ORIENTATION` column always contains the value 'ROW'.
- The `ACTION_TIMING` column contains one of the two values 'BEFORE' or 'AFTER'.
- The columns `ACTION_REFERENCE_OLD_ROW` and `ACTION_REFERENCE_NEW_ROW` contain the old and new column identifiers, respectively. This means that `ACTION_REFERENCE_OLD_ROW` always contains the value 'OLD' and `ACTION_REFERENCE_NEW_ROW` always contains the value 'NEW'.
- The `SQL_MODE` column shows the server SQL mode that was in effect at the time when the trigger was created (and thus which remains in effect for this trigger whenever it is invoked, *regardless of the current server SQL mode*). The possible range of values for this column is the same as that of the `sql_mode` system variable. See [Section 5.2.6, “SQL Modes”](#).
- The `DEFINER` column was added in MySQL 5.1.2. `DEFINER` indicates who defined the trigger.
- `CHARACTER_SET_CLIENT` is the session value of the `character_set_client` system variable when the trigger was created. `COLLATION_CONNECTION` is the session value of the `collation_connection` system variable when the trigger was created. `DATABASE_COLLATION` is the collation of the database with which the trigger is associated. These columns were added in MySQL 5.1.21.
- The following columns currently always contain NULL: `TRIGGER_CATALOG`, `EVENT_OBJECT_CATALOG`, `ACTION_CONDITION`, `ACTION_REFERENCE_OLD_TABLE`, `ACTION_REFERENCE_NEW_TABLE`, and `CREATED`.

Example, using the `ins_sum` trigger defined in [Section 20.3, “Using Triggers”](#):

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS\G
***** 1. row *****
      TRIGGER_CATALOG: NULL
      TRIGGER_SCHEMA: test
      TRIGGER_NAME: ins_sum
      EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: NULL
      EVENT_OBJECT_SCHEMA: test
      EVENT_OBJECT_TABLE: account
      ACTION_ORDER: 0
      ACTION_CONDITION: NULL
      ACTION_STATEMENT: SET @sum = @sum + NEW.amount
      ACTION_ORIENTATION: ROW
      ACTION_TIMING: BEFORE
      ACTION_REFERENCE_OLD_TABLE: NULL
      ACTION_REFERENCE_NEW_TABLE: NULL
      ACTION_REFERENCE_OLD_ROW: OLD
```



```

ACTION_REFERENCE_NEW_ROW: NEW
CREATED: NULL
SQL_MODE:
DEFINER: me@localhost

```

See also [Section 12.5.4.30, “SHOW TRIGGERS Syntax”](#).

23.17. The `INFORMATION_SCHEMA PLUGINS` Table

The `PLUGINS` table provides information about server plugins.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>PLUGIN_NAME</code>	Name	MySQL extension
<code>PLUGIN_VERSION</code>		MySQL extension
<code>PLUGIN_STATUS</code>	Status	MySQL extension
<code>PLUGIN_TYPE</code>	Type	MySQL extension
<code>PLUGIN_TYPE_VERSION</code>		MySQL extension
<code>PLUGIN_LIBRARY</code>	Library	MySQL extension
<code>PLUGIN_LIBRARY_VERSION</code>		MySQL extension
<code>PLUGIN_AUTHOR</code>		MySQL extension
<code>PLUGIN_DESCRIPTION</code>		MySQL extension
<code>PLUGIN_LICENSE</code>		MySQL extension

Notes:

- The `PLUGINS` table is a non-standard table. It was added in MySQL 5.1.5.
- The `PLUGIN_LICENSE` column was added in MySQL 5.1.12.

See also [Section 12.5.4.21, “SHOW PLUGINS Syntax”](#).

23.18. The `INFORMATION_SCHEMA ENGINES` Table

The `PLUGINS` table provides information about storage engines.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>ENGINE</code>	Engine	MySQL extension
<code>SUPPORT</code>	Support	MySQL extension
<code>COMMENT</code>	Comment	MySQL extension
<code>TRANSACTIONS</code>	Transactions	MySQL extension
<code>XA</code>	XA	MySQL extension
<code>SAVEPOINTS</code>	Savepoints	MySQL extension

Notes:

- The `ENGINES` table is a non-standard table. It was added in MySQL 5.1.5.

See also [Section 12.5.4.14, “SHOW ENGINES Syntax”](#).

23.19. The `INFORMATION_SCHEMA PARTITIONS` Table

The `PARTITIONS` table provides information about table partitions. See [Chapter 17, Partitioning](#), for more information about partitioning tables.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>TABLE_CATALOG</code>		MySQL extension
<code>TABLE_SCHEMA</code>		MySQL extension
<code>TABLE_NAME</code>		MySQL extension
<code>PARTITION_NAME</code>		MySQL extension
<code>SUBPARTITION_NAME</code>		MySQL extension
<code>PARTITION_ORDINAL_POSITION</code>		MySQL extension
<code>SUBPARTITION_ORDINAL_POSITION</code>		MySQL extension
<code>PARTITION_METHOD</code>		MySQL extension
<code>SUBPARTITION_METHOD</code>		MySQL extension
<code>PARTITION_EXPRESSION</code>		MySQL extension
<code>SUBPARTITION_EXPRESSION</code>		MySQL extension
<code>PARTITION_DESCRIPTION</code>		MySQL extension
<code>TABLE_ROWS</code>		MySQL extension
<code>AVG_ROW_LENGTH</code>		MySQL extension
<code>DATA_LENGTH</code>		MySQL extension
<code>MAX_DATA_LENGTH</code>		MySQL extension
<code>INDEX_LENGTH</code>		MySQL extension
<code>DATA_FREE</code>		MySQL extension
<code>CREATE_TIME</code>		MySQL extension
<code>UPDATE_TIME</code>		MySQL extension
<code>CHECK_TIME</code>		MySQL extension
<code>CHECKSUM</code>		MySQL extension
<code>PARTITION_COMMENT</code>		MySQL extension
<code>NODEGROUP</code>		MySQL extension
<code>TABLESPACE_NAME</code>		MySQL extension

Notes:

- The `PARTITIONS` table is a non-standard table. It was added in MySQL 5.1.6.
Each record in this table corresponds to an individual partition or subpartition of a partitioned table.
- `TABLE_CATALOG`: This column is always `NULL`.
- `TABLE_SCHEMA`: This column contains the name of the database to which the table belongs.
- `TABLE_NAME`: This column contains the name of the table containing the partition.
- `PARTITION_NAME`: The name of the partition.
- `SUBPARTITION_NAME`: If the `PARTITIONS` table record represents a subpartition, then this column contains the name of subpartition; otherwise it is `NULL`.
- `PARTITION_ORDINAL_POSITION`: All partitions are indexed in the same order as they are defined, with `1` being the number assigned to the first partition. The indexing can change as partitions are added, dropped, and reorganized; the number shown in this column reflects the current order, taking into account any indexing changes.
- `SUBPARTITION_ORDINAL_POSITION`: Subpartitions within a given partition are also indexed and reindexed in the same man-

ner as partitions are indexed within a table.

- **PARTITION_METHOD**: One of the values [RANGE](#), [LIST](#), [HASH](#), [LINEAR HASH](#), [KEY](#), or [LINEAR KEY](#); that is, one of the available partitioning types as discussed in [Section 17.2, “Partition Types”](#).
- **SUBPARTITION_METHOD**: One of the values [HASH](#), [LINEAR HASH](#), [KEY](#), or [LINEAR KEY](#); that is, one of the available subpartitioning types as discussed in [Section 17.2.5, “Subpartitioning”](#).
- **PARTITION_EXPRESSION**: This is the expression for the partitioning function used in the [CREATE TABLE](#) or [ALTER TABLE](#) statement that created the table's current partitioning scheme.

For example, consider a partitioned table created in the `test` database using this statement:

```
CREATE TABLE tp (
  c1 INT,
  c2 INT,
  c3 VARCHAR(25)
)
PARTITION BY HASH(c1 + c2)
PARTITIONS 4;
```

The **PARTITION_EXPRESSION** column in a **PARTITIONS** table record for a partition from this table displays `c1 + c2`, as shown here:

```
mysql> SELECT DISTINCT PARTITION_EXPRESSION
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_NAME='tp' AND TABLE_SCHEMA='test';
+-----+
| PARTITION_EXPRESSION |
+-----+
| c1 + c2              |
+-----+
1 row in set (0.09 sec)
```

- **SUBPARTITION_EXPRESSION**: This works in the same fashion for the subpartitioning expression that defines the subpartitioning for a table as **PARTITION_EXPRESSION** does for the partitioning expression used to define a table's partitioning.

If the table has no subpartitions, then this column is **NULL**.

- **PARTITION_DESCRIPTION**: This column is used for [RANGE](#) and [LIST](#) partitions. For a [RANGE](#) partition, it contains the value set in the partition's **VALUES LESS THAN** clause, which can be either an integer or [MAXVALUE](#). For a [LIST](#) partition, this column contains the values defined in the partition's **VALUES IN** clause, which is a comma-separated list of integer values.

For partitions whose **PARTITION_METHOD** is other than [RANGE](#) or [LIST](#), this column is always **NULL**.

- **TABLE_ROWS**: The number of table rows in the partition.
- **AVG_ROW_LENGTH**: The average length of the rows stored in this partition or subpartition, in bytes.

This is the same as **DATA_LENGTH** divided by **TABLE_ROWS**.

- **DATA_LENGTH**: The total length of all rows stored in this partition or subpartition, in bytes — that is, the total number of bytes stored in the partition or subpartition.
- **MAX_DATA_LENGTH**: The maximum number of bytes that can be stored in this partition or subpartition.
- **INDEX_LENGTH**: The length of the index file for this partition or subpartition, in bytes.
- **DATA_FREE**: The number of bytes allocated to the partition or subpartition but not used.
- **CREATE_TIME**: The time of the partition's or subpartition's creation.
- **UPDATE_TIME**: The time that the partition or subpartition was last modified.
- **CHECK_TIME**: The last time that the table to which this partition or subpartition belongs was checked.

Note

Some storage engines do not update this time; for tables using these storage engines, this value is always **NULL**.

- **CHECKSUM**: The checksum value, if any; otherwise, this column is `NULL`.
- **PARTITION_COMMENT**: This column contains the text of any comment made for the partition.

The default value for this column is an empty string.

- **NODEGROUP**: This is the nodegroup to which the partition belongs. This is relevant only to MySQL Cluster tables; otherwise the value of this column is always `0`.
- **TABLESPACE_NAME**: This column contains the name of tablespace to which the partition belongs. In MySQL 5.1, the value of this column is always `DEFAULT`.
-

Important

If any partitioned tables created in a MySQL version prior to MySQL 5.1.6 are present following an upgrade to MySQL 5.1.6 or later, it is not possible to `SELECT` from, `SHOW`, or `DESCRIBE` the `PARTITIONS` table. See [Section C.1.24, “Changes in release 5.1.6 \(01 February 2006\)”](#) before upgrading from MySQL 5.1.5 or earlier to MySQL 5.1.6 or later.

- A non-partitioned table has one record in `INFORMATION_SCHEMA.PARTITIONS`; however, the values of the `PARTITION_NAME`, `SUBPARTITION_NAME`, `PARTITION_ORDINAL_POSITION`, `SUBPARTITION_ORDINAL_POSITION`, `PARTITION_METHOD`, `SUBPARTITION_METHOD`, `PARTITION_EXPRESSION`, `SUBPARTITION_EXPRESSION`, and `PARTITION_DESCRIPTION` columns are all `NULL`. (The `PARTITION_COMMENT` column in this case is blank.)

In MySQL 5.1, there is also only one record in the `PARTITIONS` table for a table using the `NDBCluster` storage engine. The same columns are also `NULL` (or empty) as for a non-partitioned table.

23.20. The `INFORMATION_SCHEMA EVENTS` Table

The `EVENTS` table provides information about scheduled events, which are discussed in [Chapter 21, Event Scheduler](#).

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>EVENT_CATALOG</code>		<code>NULL</code> , MySQL extension
<code>EVENT_SCHEMA</code>	<code>Db</code>	MySQL extension
<code>EVENT_NAME</code>	<code>Name</code>	MySQL extension
<code>DEFINER</code>	<code>Definer</code>	MySQL extension
<code>TIME_ZONE</code>	<code>Time zone</code>	MySQL extension
<code>EVENT_BODY</code>		MySQL extension
<code>EVENT_DEFINITION</code>		MySQL extension
<code>EVENT_TYPE</code>	<code>Type</code>	MySQL extension
<code>EXECUTE_AT</code>	<code>Execute at</code>	MySQL extension
<code>INTERVAL_VALUE</code>	<code>Interval value</code>	MySQL extension
<code>INTERVAL_FIELD</code>	<code>Interval field</code>	MySQL extension
<code>SQL_MODE</code>		MySQL extension
<code>STARTS</code>	<code>Starts</code>	MySQL extension
<code>ENDS</code>	<code>Ends</code>	MySQL extension
<code>STATUS</code>	<code>Status</code>	MySQL extension
<code>ON_COMPLETION</code>		MySQL extension
<code>CREATED</code>		MySQL extension
<code>LAST_ALTERED</code>		MySQL extension
<code>LAST_EXECUTED</code>		MySQL extension
<code>EVENT_COMMENT</code>		MySQL extension
<code>ORIGINATOR</code>	<code>Originator</code>	MySQL extension
<code>CHARACTER_SET_CLIENT</code>		MySQL extension

COLLATION_CONNECTION		MySQL extension
DATABASE_COLLATION		MySQL extension

Notes:

- The **EVENTS** table is a non-standard table. It was added in MySQL 5.1.6.
- **EVENT_CATALOG**: The value of this column is always **NULL**.
- **EVENT_SCHEMA**: The name of the schema (database) to which this event belongs.
- **EVENT_NAME**: The name of the event.
- **DEFINER**: The user who created the event. Always displayed in '*user_name*'@'*host_name*' format.
- **TIME_ZONE**: The time zone in effect when schedule for the event was last modified. If the event's schedule has not been modified since the event was created, then this is the time zone that was in effect at the event's creation. The default value is **SYSTEM**.

This column was added in MySQL 5.1.17. See [Section C.1.10, “Changes in release 5.1.17 \(04 April 2007\)”](#), for important information if you are using the Event Scheduler and are upgrading from MySQL 5.1.16 (or earlier) to MySQL 5.1.17 (or later).

- **EVENT_BODY**: The language used for the statements in the event's **DO** clause; in MySQL 5.1, this is always **SQL**.

This column was added in MySQL 5.1.12. It is not to be confused with the column of the same name (now named **EVENT_DEFINITION**) that existed in earlier MySQL versions.

- **EVENT_DEFINITION**: The text of the SQL statement making up the event's **DO** clause; in other words, the statement executed by this event.

Note

Prior to MySQL 5.1.12, this column was named **EVENT_BODY**.

- **EVENT_TYPE**: One of the two values **ONE TIME** or **RECURRING**.
- **EXECUTE_AT**: For a one-time event, this is the **DATETIME** value specified in the **AT** clause of the **CREATE EVENT** statement used to create the event, or of the last **ALTER EVENT** statement that modified the event. The value shown in this column reflects the addition or subtraction of any **INTERVAL** value included in the event's **AT** clause. For example, if an event is created using **ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR**, and the event was created at 2006-02-09 14:05:30, the value shown in this column would be '**2006-02-10 20:05:30**'.

If the event's timing is determined by an **EVERY** clause instead of an **AT** clause (that is, if the event is recurring), the value of this column is **NULL**.

- **INTERVAL_VALUE**: For recurring events, this column contains the numeric portion of the event's **EVERY** clause.

For a one-time event (that is, an event whose timing is determined by an **AT** clause), this column's value is **NULL**.

- **INTERVAL_FIELD**: For recurring events, this column contains the units portion of the **EVERY** clause governing the timing of the event, prefixed with '**INTERVAL_**'. Thus, this column contains a value such as '**INTERVAL_YEAR**', '**INTERVAL_QUARTER**', '**INTERVAL_DAY**', and so on.

For a one-time event (that is, an event whose timing is determined by an **AT** clause), this column's value is **NULL**.

- **SQL_MODE**: The SQL mode in effect at the time the event was created or altered.
- **STARTS**: For a recurring event whose definition includes a **STARTS** clause, this column contains the corresponding **DATETIME** value. As with the **EXECUTE_AT** column, this value resolves any expressions used.

If there is no **STARTS** clause affecting the timing of the event, this column is empty. (Prior to MySQL 5.1.8, it contained **NULL** in such cases.)

- **ENDS**: For a recurring event whose definition includes a **ENDS** clause, this column contains the corresponding **DATETIME** value.

As with the `EXECUTE_AT` column (see previous example), this value resolves any expressions used.

If there is no `ENDS` clause affecting the timing of the event, this column contains `NULL`.

- **STATUS:** One of the three values `ENABLED`, `DISABLED`, or `SLAVESIDE_DISABLED`.

`SLAVESIDE_DISABLED` was added to the list of possible values for this column in MySQL 5.1.18. This value indicates that the creation of the event occurred on another MySQL server acting as a replication master and was replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave. See [Section 15.3.1.5, “Replication of Invoked Features”](#), for more information.

- **ON_COMPLETION:** One of the two values `PRESERVE` or `NOT PRESERVE`.
- **CREATED:** The date and time when the event was created. This is a `DATETIME` value.
- **LAST_ALTERED:** The date and time when the event was last modified. This is a `DATETIME` value. If the event has not been modified since its creation, this column holds the same value as the `CREATED` column.
- **LAST_EXECUTED:** The date and time when the event last executed. A `DATETIME` value. If the event has never executed, this column's value is `NULL`.
- **EVENT_COMMENT:** The text of a comment, if the event has one. If there is no comment, the value of this column is an empty string.
- **ORIGINATOR:** The server ID of the MySQL server on which the event was created; used in replication. The default value is 0. This column was added in MySQL 5.1.18.
- **CHARACTER_SET_CLIENT** is the session value of the `character_set_client` system variable when the event was created. **COLLATION_CONNECTION** is the session value of the `collation_connection` system variable when the event was created. **DATABASE_COLLATION** is the collation of the database with which the event is associated. These columns were added in MySQL 5.1.21.

Example: Suppose the user `jon@ghidora` creates an event named `e_daily`, and then modifies it a few minutes later using an `ALTER EVENT` statement, as shown here:

```
DELIMITER |
CREATE EVENT e_daily
ON SCHEDULE EVERY 1 DAY
STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
DISABLE
COMMENT 'Saves total number of sessions and
        clears the table once per day.'
DO
BEGIN
    INSERT INTO site_activity.totals (when, total)
    SELECT CURRENT_TIMESTAMP, COUNT(*)
    FROM site_activity.sessions;
    DELETE FROM site_activity.sessions;
END |
DELIMITER ;
ALTER EVENT e_daily
ENABLED;
```

(Note that comments can span multiple lines.)

This user can then run the following `SELECT` statement, and obtain the output shown:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
> WHERE EVENT_NAME = 'e_daily'
> AND EVENT_SCHEMA = 'myschema'\G
***** 1. row *****
EVENT_CATALOG: NULL
EVENT_SCHEMA: myschema
EVENT_NAME: e_daily
DEFINER: jon@ghidora
EVENT_BODY: BEGIN
    INSERT INTO site_activity.totals (when, total)
    SELECT CURRENT_TIMESTAMP, COUNT(*)
    FROM site_activity.sessions;
    DELETE FROM site_activity.sessions;
END
```

```

EVENT_TYPE: RECURRING
EXECUTE_AT: NULL
INTERVAL_VALUE: 1
INTERVAL_FIELD: INTERVAL_DAY
SQL_MODE: NULL
STARTS: 2006-02-09 10:41:23
ENDS: NULL
STATUS: ENABLED
ON_COMPLETION: DROP
CREATED: 2006-02-09 14:35:35
LAST_ALTERED: 2006-02-09 14:41:23
LAST_EXECUTED: NULL
EVENT_COMMENT: Saves total number of sessions and
                clears the table once per day.
ORIGINATOR: 0
1 row in set (0.50 sec)

```

Prior to MySQL 5.1.17, the times displayed in the `STARTS`, `ENDS`, and `LAST_EXECUTED` columns were given in terms of Universal Time (GMT or UTC), regardless of the server's time zone setting ([Bug#16420](#)). Beginning with MySQL 5.1.17, these times are all given in terms of local time as determined by the MySQL server's `time_zone` setting. (The same was true of the `starts`, `ends`, and `last_executed` columns of the `mysql.event` table as well as the `Starts` and `Ends` columns in the output of `SHOW [FULL] EVENTS`.)

The `CREATED` and `LAST_ALTERED` columns use the server time zone (as do the `created` and `last_altered` columns of the `mysql.event` table).

See also [Section 12.5.4.16](#), “`SHOW EVENTS`”.

23.21. The `INFORMATION_SCHEMA FILES` Table

The `FILES` table provides information about the files in which MySQL `NDB` Disk Data tables are stored.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>FILE_ID</code>		MySQL extension
<code>FILE_NAME</code>		MySQL extension
<code>FILE_TYPE</code>		MySQL extension
<code>TABLESPACE_NAME</code>		MySQL extension
<code>TABLE_CATALOG</code>		MySQL extension
<code>TABLE_SCHEMA</code>		MySQL extension
<code>TABLE_NAME</code>		MySQL extension
<code>LOGFILE_GROUP_NAME</code>		MySQL extension
<code>LOGFILE_GROUP_NUMBER</code>		MySQL extension
<code>ENGINE</code>		MySQL extension
<code>FULLTEXT_KEYS</code>		MySQL extension
<code>DELETED_ROWS</code>		MySQL extension
<code>UPDATE_COUNT</code>		MySQL extension
<code>FREE_EXTENTS</code>		MySQL extension
<code>TOTAL_EXTENTS</code>		MySQL extension
<code>EXTENT_SIZE</code>		MySQL extension
<code>INITIAL_SIZE</code>		MySQL extension
<code>MAXIMUM_SIZE</code>		MySQL extension
<code>AUTOEXTEND_SIZE</code>		MySQL extension
<code>CREATION_TIME</code>		MySQL extension
<code>LAST_UPDATE_TIME</code>		MySQL extension
<code>LAST_ACCESS_TIME</code>		MySQL extension
<code>RECOVER_TIME</code>		MySQL extension
<code>TRANSACTION_COUNTER</code>		MySQL extension

VERSION		MySQL extension
ROW_FORMAT		MySQL extension
TABLE_ROWS		MySQL extension
AVG_ROW_LENGTH		MySQL extension
DATA_LENGTH		MySQL extension
MAX_DATA_LENGTH		MySQL extension
INDEX_LENGTH		MySQL extension
DATA_FREE		MySQL extension
CREATE_TIME		MySQL extension
UPDATE_TIME		MySQL extension
CHECK_TIME		MySQL extension
CHECKSUM		MySQL extension
STATUS		MySQL extension
EXTRA		MySQL extension

Notes:

- **FILE_ID** column values are auto-generated.
- **FILE_NAME** is the name of an **UNDO** log file created by **CREATE LOGFILE GROUP** or **ALTER LOGFILE GROUP**, or of a data file created by **CREATE TABLESPACE** or **ALTER TABLESPACE**.
- **FILE_TYPE** is one of the values **UNDOFILE** or **DATAFILE**.
- **TABLESPACE_NAME** is the name of the tablespace with which the file is associated.
- In MySQL 5.1, the value of the **TABLESPACE_CATALOG** column is always **NULL**.
- **TABLE_NAME** is the name of the Disk Data table with which the file is associated, if any.
- The **LOGFILE_GROUP_NAME** column gives the name of the log file group to which the log file or data file belongs.
- For an **UNDO** log file, the **LOGFILE_GROUP_NUMBER** contains the auto-generated ID number of the log file group to which the log file belongs.
- For a MySQL Cluster Disk Data log file or data file, the value of the **ENGINE** column is always **NDB** or **NDBCLUSTER**.
- For a MySQL Cluster Disk Data log file or data file, the value of the **FULLTEXT_KEYS** column is always empty.
- The **FREE_EXTENTS** column displays the number of extents which have not yet been used by the file. The **TOTAL_EXTENTS** column show the total number of extents allocated to the file.

The difference between these two columns is the number of extents currently in use by the file:

```
SELECT TOTAL_EXTENTS - FREE_EXTENTS AS extents_used
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

You can approximate the amount of disk space in use by the file by multiplying this difference by the value of the **EXTENT_SIZE** column, which gives the size of an extent for the file in bytes:

```
SELECT (TOTAL_EXTENTS - FREE_EXTENTS) * EXTENT_SIZE AS bytes_used
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

Similarly, you can estimate the amount of space that remains available in a given file by multiplying **FREE_EXTENTS** by **EXTENT_SIZE**:


```
SELECT FREE_EXTENTS * EXTENT_SIZE AS bytes_free
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

Important

The byte values produced by the preceding queries are approximations only, and their precision is inversely proportional to the value of `EXTENT_SIZE`. That is, the larger `EXTENT_SIZE` becomes, the less accurate the approximations are.

It is also important to remember that once an extent is used, it cannot be freed again without dropping the data file of which it is a part. This means that deletes from a Disk Data table do *not* release disk space.

The extent size can be set in a `CREATE TABLESPACE` statement. See [Section 12.1.11, “CREATE TABLESPACE Syntax”](#), for more information.

- The `INITIAL_SIZE` column shows the size in bytes of the file. This is the same value that was used in the `INITIAL_SIZE` clause of the `CREATE LOGFILE GROUP`, `ALTER LOGFILE GROUP`, `CREATE TABLESPACE`, or `ALTER TABLESPACE` statement used to create the file.

For MySQL 5.1 Cluster Disk Data files, the value of the `MAXIMUM_SIZE` column is always the same as `INITIAL_SIZE`, and the `AUTOEXTEND_SIZE` column is always empty.

- The `CREATION_TIME` column shows the date and time when the file was created. The `LAST_UPDATE_TIME` column displays the date and time when the file was last modified. The `LAST_ACCESSED` column provides the date and time when the file was last accessed by the server.

Currently, the values of these columns are as reported by the operating system, and are not supplied by the NDB storage engine. Where no value is provided by the operating system, these columns display `0000-00-00 00:00:00`.

- For MySQL Cluster Disk Data files, the value of the `RECOVER_TIME` and `TRANSACTION_COUNTER` columns is always 0.
- For MySQL 5.1 Cluster Disk Data files, the following columns are always `NULL`:
 - `VERSION`
 - `ROW_FORMAT`
 - `TABLE_ROWS`
 - `AVG_ROW_LENGTH`
 - `DATA_LENGTH`
 - `MAX_DATA_LENGTH`
 - `INDEX_LENGTH`
 - `DATA_FREE`
 - `CREATE_TIME`
 - `UPDATE_TIME`
 - `CHECK_TIME`
 - `CHECKSUM`
- For MySQL Cluster Disk Data files, the value of the `STATUS` column is always `NORMAL`.
- For MySQL Cluster Disk Data files, the `EXTRA` column shows which data node the file belongs to, as each data node has its own copy of the file. For example, suppose you use this statement on a MySQL Cluster with four data nodes:

```
CREATE LOGFILE GROUP mygroup
ADD UNDOFILE 'new_undo.dat'
INITIAL_SIZE 2G
ENGINE NDB;
```

After running the `CREATE LOGFILE GROUP` statement successfully, you should see a result similar to the one shown here for this query against the `FILES` table:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_TYPE, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE FILE_NAME = 'new_undo.dat';
```

LOGFILE_GROUP_NAME	FILE_TYPE	EXTRA
mygroup	UNDO FILE	CLUSTER_NODE=3
mygroup	UNDO FILE	CLUSTER_NODE=4
mygroup	UNDO FILE	CLUSTER_NODE=5
mygroup	UNDO FILE	CLUSTER_NODE=6

4 rows in set (0.01 sec)

- The `FILES` table is a non-standard table. It was added in MySQL 5.1.6.
- Beginning with MySQL 5.1.14, an additional row is present in the `FILES` table following the creation of a logfile group. This row has `NULL` for the value of the `FILE_NAME` column. For this row, the value of the `FILE_ID` column is always `0`, that of the `FILE_TYPE` column is always `UNDO FILE`, and that of the `STATUS` column is always `NORMAL`. In MySQL 5.1 the value of the `ENGINE` column is always `ndbcluster`.

This row shows in the `FREE_EXTENTS` column the total number of free extents available to all undo files belonging to a given log file group whose name and number are shown in the `LOGFILE_GROUP_NAME` and `LOGFILE_GROUP_NUMBER` columns, respectively.

Suppose there are no existing log file groups on your MySQL Cluster, and you create one using the following statement:

```
mysql> CREATE LOGFILE GROUP lg1
-> ADD UNDOFILE 'undofile.dat'
-> INITIAL_SIZE = 16M
-> UNDO_BUFFER_SIZE = 1M
-> ENGINE = NDB;
```

Query OK, 0 rows affected (3.81 sec)

You can now see this `NULL` row when you query the `FILES` table:

```
mysql> SELECT DISTINCT
-> FILE_NAME AS File,
-> FREE_EXTENTS AS Free,
-> TOTAL_EXTENTS AS Total,
-> EXTENT_SIZE AS Size,
-> INITIAL_SIZE AS Initial
-> FROM INFORMATION_SCHEMA.FILES;
```

File	Free	Total	Size	Initial
undofile.dat	NULL	4194304	4	16777216
NULL	4184068	NULL	4	NULL

2 rows in set (0.01 sec)

The total number of free extents available for undo logging is always somewhat less than the sum of the `TOTAL_EXTENTS` column values for all undo files in the log file group due to overhead required for maintaining the undo files. This can be seen by adding a second undo file to the log file group, then repeating the previous query against the `FILES` table:

```
mysql> ALTER LOGFILE GROUP lg1
-> ADD UNDOFILE 'undofile02.dat'
-> INITIAL_SIZE = 4M
-> ENGINE = NDB;
```

Query OK, 0 rows affected (1.02 sec)

```
mysql> SELECT DISTINCT
-> FILE_NAME AS File,
-> FREE_EXTENTS AS Free,
-> TOTAL_EXTENTS AS Total,
-> EXTENT_SIZE AS Size,
-> INITIAL_SIZE AS Initial
-> FROM INFORMATION_SCHEMA.FILES;
```

File	Free	Total	Size	Initial
undofile.dat	NULL	4194304	4	16777216
undofile02.dat	NULL	1048576	4	4194304
NULL	5223944	NULL	4	NULL

```
3 rows in set (0.01 sec)
```

The amount of free space in bytes which is available for undo logging by Disk Data tables using this log file group can be approximated by multiplying the number of free extents by the initial size:

```
mysql> SELECT
->   FREE_EXTENTS AS 'Free Extents',
->   FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE LOGFILE_GROUP_NAME = 'lg1'
-> AND FILE_NAME IS NULL;
+-----+-----+
| Free Extents | Free Bytes |
+-----+-----+
|      5223944 |  20895776 |
+-----+-----+
1 row in set (0.02 sec)
```

If you create a Disk Data table and then insert some rows into it, you can see approximately how much space remains for undo logging afterwards, for example:

```
mysql> CREATE TABLESPACE ts1
->   ADD DATAFILE 'data1.dat'
->   USE LOGFILE GROUP lg1
->   INITIAL_SIZE 512M
->   ENGINE = NDB;
Query OK, 0 rows affected (8.71 sec)

mysql> CREATE TABLE dd (
->   c1 INT NOT NULL PRIMARY KEY,
->   c2 INT,
->   c3 DATE
-> )
-> TABLESPACE ts1 STORAGE DISK
-> ENGINE = NDB;
Query OK, 0 rows affected (2.11 sec)

mysql> INSERT INTO dd VALUES
->   (NULL, 1234567890, '2007-02-02'),
->   (NULL, 1126789005, '2007-02-03'),
->   (NULL, 1357924680, '2007-02-04'),
->   (NULL, 1642097531, '2007-02-05');
Query OK, 4 rows affected (0.01 sec)

mysql> SELECT
->   FREE_EXTENTS AS 'Free Extents',
->   FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE LOGFILE_GROUP_NAME = 'lg1'
-> AND FILE_NAME IS NULL;
+-----+-----+
| Free Extents | Free Bytes |
+-----+-----+
|      5207565 |  20830260 |
+-----+-----+
1 row in set (0.01 sec)
```

- There are no `SHOW` commands associated with the `FILES` table.
- For additional examples using the `FILES` table to obtain information about Cluster Disk Data tables, see [Section 16.11, “MySQL Cluster Disk Data Tables”](#).

23.22. The INFORMATION_SCHEMA PROCESSLIST Table

The `PROCESSLIST` table provides information about which threads are running.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
ID	Id	MySQL extension
USER	User	MySQL extension
HOST	Host	MySQL extension
DB	db	MySQL extension
COMMAND	Command	MySQL extension

<code>TIME</code>	<code>Time</code>	MySQL extension
<code>STATE</code>	<code>State</code>	MySQL extension
<code>INFO</code>	<code>Info</code>	MySQL extension

For an extensive description of the table columns, see [Section 12.5.4.25, “SHOW PROCESSLIST Syntax”](#).

Notes:

- The `PROCESSLIST` table is a non-standard table. It was added in MySQL 5.1.7.
- Like the output from the corresponding `SHOW` statement, the `PROCESSLIST` table will only show information about your own threads, unless you have the `PROCESS` privilege, in which case you will see information about other threads, too. As an anonymous user, you cannot see any rows at all.
- If an SQL statement refers to `INFORMATION_SCHEMA.PROCESSLIST`, then MySQL will populate the entire table once, when statement execution begins, so there is read consistency during the statement. There is no read consistency for a multi-statement transaction, though.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
SHOW FULL PROCESSLIST
```

23.23. The `INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS` Table

The `REFERENTIAL_CONSTRAINTS` table provides information about foreign keys.

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>CONSTRAINT_CATALOG</code>		NULL
<code>CONSTRAINT_SCHEMA</code>		
<code>CONSTRAINT_NAME</code>		
<code>UNIQUE_CONSTRAINT_CATALOG</code>		NULL
<code>UNIQUE_CONSTRAINT_SCHEMA</code>		
<code>UNIQUE_CONSTRAINT_NAME</code>		
<code>MATCH_OPTION</code>		
<code>UPDATE_RULE</code>		
<code>DELETE_RULE</code>		
<code>TABLE_NAME</code>		
<code>REFERENCED_TABLE_NAME</code>		

Notes:

- The `REFERENTIAL_CONSTRAINTS` table was added in MySQL 5.1.10. The `REFERENCED_TABLE_NAME` column was added in MySQL 5.1.16.
- `TABLE_NAME` has the same value as `TABLE_NAME` in `INFORMATION_SCHEMA.TABLE_CONSTRAINTS`.
- `CONSTRAINT_SCHEMA` and `CONSTRAINT_NAME` identify the foreign key.
- `UNIQUE_CONSTRAINT_SCHEMA`, `UNIQUE_CONSTRAINT_NAME`, and `REFERENCED_TABLE_NAME` identify the referenced key. (Note: Before MySQL 5.1.16, `UNIQUE_CONSTRAINT_NAME` incorrectly named the referenced table, not the constraint.)
- The only valid value at this time for `MATCH_OPTION` is `NONE`.

- The possible values for `UPDATE_RULE` or `DELETE_RULE` are `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT`, `NO ACTION`.

23.24. The `INFORMATION_SCHEMA GLOBAL_STATUS` and `SESSION_STATUS` Tables

The `GLOBAL_STATUS` and `SESSION_STATUS` tables provide information about server status variables. Their contents correspond to the information produced by the `SHOW GLOBAL STATUS` and `SHOW SESSION STATUS` statements (see [Section 12.5.4.27](#), “`SHOW STATUS Syntax`”).

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>VARIABLE_NAME</code>	<code>Variable_name</code>	
<code>VARIABLE_VALUE</code>	Value	

Notes:

- The `GLOBAL_STATUS` and `SESSION_STATUS` tables were added in MySQL 5.1.12.
- Beginning with MySQL 5.1.19, the `VARIABLE_VALUE` column for each of these tables is defined as `VARCHAR(20480)`. Previously, this column had the data type `DECIMAL(22, 7)`, but was changed to avoid loss of data when working with status variables whose values were strings ([Bug#26994](#)).

23.25. The `INFORMATION_SCHEMA GLOBAL_VARIABLES` and `SESSION_VARIABLES` Tables

The `GLOBAL_VARIABLES` and `SESSION_VARIABLES` tables provide information about server status variables. Their contents correspond to the information produced by the `SHOW GLOBAL VARIABLES` and `SHOW SESSION VARIABLES` statements (see [Section 12.5.4.31](#), “`SHOW VARIABLES Syntax`”).

<code>INFORMATION_SCHEMA</code> Name	<code>SHOW</code> Name	Remarks
<code>VARIABLE_NAME</code>	<code>Variable_name</code>	
<code>VARIABLE_VALUE</code>	Value	

Notes:

- The `GLOBAL_VARIABLES` and `SESSION_VARIABLES` tables were added in MySQL 5.1.12.
- Beginning with MySQL 5.1.19, the `VARIABLE_VALUE` column for each of these tables is defined as `VARCHAR(20480)`. Previously, this column had the data type `LONGTEXT`; this was changed in order to make these tables consistent with the `GLOBAL_STATUS` and `SESSION_STATUS` tables, whose definitions had been changed in that version (see [Section 23.24](#), “[The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables](#)”).

23.26. Other `INFORMATION_SCHEMA` Tables

We intend to implement additional `INFORMATION_SCHEMA` tables. In particular, we acknowledge the need for the `PARAMETERS` table.

23.27. Extensions to `SHOW` Statements

Some extensions to `SHOW` statements accompany the implementation of `INFORMATION_SCHEMA`:

- **SHOW** can be used to get information about the structure of **INFORMATION_SCHEMA** itself.
- Several **SHOW** statements accept a **WHERE** clause that provides more flexibility in specifying which rows to display.

INFORMATION_SCHEMA is an information database, so its name is included in the output from **SHOW DATABASES**. Similarly, **SHOW TABLES** can be used with **INFORMATION_SCHEMA** to obtain a list of its tables:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+-----+
| Tables_in_INFORMATION_SCHEMA |
+-----+
| CHARACTER_SETS                |
| COLLATIONS                    |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                      |
| COLUMN_PRIVILEGES             |
| ENGINES                       |
| EVENTS                       |
| FILES                         |
| GLOBAL_STATUS                 |
| GLOBAL_VARIABLES              |
| KEY_COLUMN_USAGE              |
| PARTITIONS                    |
| PLUGINS                       |
| PROCESSLIST                   |
| REFERENTIAL_CONSTRAINTS       |
| ROUTINES                      |
| SCHEMATA                      |
| SCHEMA_PRIVILEGES             |
| SESSION_STATUS                |
| SESSION_VARIABLES             |
| STATISTICS                    |
| TABLES                       |
| TABLE_CONSTRAINTS            |
| TABLE_PRIVILEGES             |
| TRIGGERS                      |
| USER_PRIVILEGES               |
| VIEWS                         |
+-----+
27 rows in set (0.00 sec)
```

SHOW COLUMNS and **DESCRIBE** can display information about the columns in individual **INFORMATION_SCHEMA** tables.

SHOW statements that accept a **LIKE** clause to limit the rows displayed also allow a **WHERE** clause that enables specification of more general conditions that selected rows must satisfy:

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW INDEX
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW TRIGGERS
SHOW VARIABLES
```

The **WHERE** clause, if present, is evaluated against the column names displayed by the **SHOW** statement. For example, the **SHOW CHARACTER SET** statement produces these output columns:

```
mysql> SHOW CHARACTER SET;
+-----+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+-----+
| big5    | Big5 Traditional Chinese | big5_chinese_ci | 2 |
| dec8    | DEC West European | dec8_swedish_ci | 1 |
| cp850   | DOS West European | cp850_general_ci | 1 |
| hp8     | HP West European | hp8_english_ci | 1 |
| koi8r   | KOI8-R Relcom Russian | koi8r_general_ci | 1 |
| latin1  | cp1252 West European | latin1_swedish_ci | 1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1 |
| ...     | ... | ... | ... |
```

To use a **WHERE** clause with **SHOW CHARACTER SET**, you would refer to those column names. As an example, the following statement displays information about character sets for which the default collation contains the string **'japanese'**:

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
```

Charset	Description	Default collation	Maxlen
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3

This statement displays the multi-byte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
```

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
euckr	EUC-KR Korean	euckr_korean_ci	2
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
utf8	UTF-8 Unicode	utf8_general_ci	3
ucs2	UCS-2 Unicode	ucs2_general_ci	2
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3

Chapter 24. Precision Math

MySQL 5.1 provides support for precision math: numeric value handling that results in extremely accurate results and a high degree control over invalid values. Precision math is based on these two features:

- SQL modes that control how strict the server is about accepting or rejecting invalid data.
- The MySQL library for fixed-point arithmetic.

These features have several implications for numeric operations:

- **Precise calculations:** For exact-value numbers, calculations do not introduce floating-point errors. Instead, exact precision is used. For example, a number such as `.0001` is treated as an exact value rather than as an approximation, and summing it 10,000 times produces a result of exactly `1`, not a value that merely “close” to 1.
- **Well-defined rounding behavior:** For exact-value numbers, the result of `ROUND()` depends on its argument, not on environmental factors such as how the underlying C library works.
- **Platform independence:** Operations on exact numeric values are the same across different platforms such as Windows and Unix.
- **Control over handling of invalid values:** Overflow and division by zero are detectable and can be treated as errors. For example, you can treat a value that is too large for a column as an error rather than having the value truncated to lie within the range of the column's data type. Similarly, you can treat division by zero as an error rather than as an operation that produces a result of `NULL`. The choice of which approach to take is determined by the setting of the `sql_mode` system variable.

An important result of these features is that MySQL 5.1 provides a high degree of compliance with standard SQL.

The following discussion covers several aspects of how precision math works (including possible incompatibilities with older applications). At the end, some examples are given that demonstrate how MySQL 5.1 handles numeric operations precisely. For information about using the `sql_mode` system variable to control the SQL mode, see [Section 5.2.6, “SQL Modes”](#).

24.1. Types of Numeric Values

The scope of precision math for exact-value operations includes the exact-value data types (`DECIMAL` and integer types) and exact-value numeric literals. Approximate-value data types and numeric literals still are handled as floating-point numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar need not be both exact-value or both approximate-value. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types.

The `FLOAT` and `DOUBLE` data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with `FLOAT` or `DOUBLE` are `DOUBLE PRECISION` and `REAL`.

24.2. `DECIMAL` Data Type Changes

This section discusses the characteristics of the `DECIMAL` data type (and its synonyms) in MySQL 5.1, with particular regard to the following topics:

- Maximum number of digits
- Storage format

- Storage requirements
- The non-standard MySQL extension to the upper range of **DECIMAL** columns

Possible incompatibilities with applications that are written for older versions of MySQL are noted throughout this section.

The declaration syntax for a **DECIMAL** column is **DECIMAL**(*M*, *D*). The ranges of values for the arguments in MySQL 5.1 are as follows:

- *M* is the maximum number of digits (the precision). It has a range of 1 to 65. (Older versions of MySQL allowed a range of 1 to 254.)
- *D* is the number of digits to the right of the decimal point (the scale). It has a range of 0 to 30 and must be no larger than *M*.

The maximum value of 65 for *M* means that calculations on **DECIMAL** values are accurate up to 65 digits. This limit of 65 digits of precision also applies to exact-value numeric literals, so the maximum range of such literals is different from before. (In older versions of MySQL, decimal values could have up to 254 digits. However, calculations were done using floating-point and thus were approximate, not exact.)

Values for **DECIMAL** columns in MySQL 5.1 are stored using a binary format that packs nine decimal digits into four bytes. The storage requirements for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires four bytes, and any digits left over require some fraction of four bytes. For example, a **DECIMAL**(18, 9) column has nine digits on either side of the decimal point, so the integer part and the fractional part each require four bytes. A **DECIMAL**(20, 10) column has ten digits on either side of the decimal point. Each part requires four bytes for nine of the digits, and one byte for the remaining digit.

The storage required for leftover digits is given by the following table:

Leftover Digits	Number of Bytes
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4
9	4

Unlike some older versions of MySQL (prior to 5.0.3), **DECIMAL** columns in MySQL 5.1 do not store a leading + character or leading 0 digits. If you insert +0003.1 into a **DECIMAL**(5, 1) column, it is stored as 3.1. Applications that rely on the older behavior must be modified to account for this change.

DECIMAL columns in MySQL 5.1 do not allow values larger than the range implied by the column definition. For example, a **DECIMAL**(3, 0) column supports a range of -999 to 999. A **DECIMAL**(*M*, *D*) column allows at most *M* - *D* digits to the left of the decimal point. This is not compatible with applications relying on older versions of MySQL that allowed storing an extra digit in lieu of a + sign.

The SQL standard requires that the precision of **NUMERIC**(*M*, *D*) be exactly *M* digits. For **DECIMAL**(*M*, *D*), the standard requires a precision of at least *M* digits but allows more. In MySQL, **DECIMAL**(*M*, *D*) and **NUMERIC**(*M*, *D*) are the same, and both have a precision of exactly *M* digits.

For more detailed information about porting applications that rely on the old treatment of the **DECIMAL** data type, see the *MySQL 5.0 Reference Manual*.

24.3. Expression Handling

With precision math, exact-value numbers are used as given whenever possible. For example, numbers in comparisons are used exactly as given without a change in value. In strict SQL mode, for [INSERT](#) into a column with an exact data type ([DECIMAL](#) or integer), a number is inserted with its exact value if it is within the column range. When retrieved, the value should be the same as what was inserted. (Without strict mode, truncation for [INSERT](#) is allowable.)

Handling of a numeric expression depends on what kind of values the expression contains:

- If any approximate values are present, the expression is approximate and is evaluated using floating-point arithmetic.
- If no approximate values are present, the expression contains only exact values. If any exact value contains a fractional part (a value following the decimal point), the expression is evaluated using [DECIMAL](#) exact arithmetic and has a precision of 65 digits. (The term “exact” is subject to the limits of what can be represented in binary. For example, $1.0/3.0$ can be approximated in decimal notation as $.333\dots$, but not written as an exact number, so $(1.0/3.0)*3.0$ does not evaluate to exactly 1.0 .)
- Otherwise, the expression contains only integer values. The expression is exact and is evaluated using integer arithmetic and has a precision the same as [BIGINT](#) (64 bits).

If a numeric expression contains any strings, they are converted to double-precision floating-point values and the expression is approximate.

Inserts into numeric columns are affected by the SQL mode, which is controlled by the `sql_mode` system variable. (See [Section 5.2.6, “SQL Modes”](#).) The following discussion mentions strict mode (selected by the `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` mode values) and `ERROR_FOR_DIVISION_BY_ZERO`. To turn on all restrictions, you can simply use `TRADITIONAL` mode, which includes both strict mode values and `ERROR_FOR_DIVISION_BY_ZERO`:

```
mysql> SET sql_mode='TRADITIONAL';
```

If a number is inserted into an exact type column ([DECIMAL](#) or integer), it is inserted with its exact value if it is within the column range.

If the value has too many digits in the fractional part, rounding occurs and a warning is generated. Rounding is done as described in [Section 24.4, “Rounding Behavior”](#).

If the value has too many digits in the integer part, it is too large and is handled as follows:

- If strict mode is not enabled, the value is truncated to the nearest legal value and a warning is generated.
- If strict mode is enabled, an overflow error occurs.

Underflow is not detected, so underflow handling is undefined.

By default, division by zero produces a result of [NULL](#) and no warning. With the `ERROR_FOR_DIVISION_BY_ZERO` SQL mode enabled, MySQL handles division by zero differently:

- If strict mode is not enabled, a warning occurs.
- If strict mode is enabled, inserts and updates involving division by zero are prohibited, and an error occurs.

In other words, inserts and updates involving expressions that perform division by zero can be treated as errors, but this requires `ERROR_FOR_DIVISION_BY_ZERO` in addition to strict mode.

Suppose that we have this statement:

```
INSERT INTO t SET i = 1/0;
```

This is what happens for combinations of strict and `ERROR_FOR_DIVISION_BY_ZERO` modes:

<code>sql_mode</code> Value	Result
' ' (Default)	No warning, no error; <code>i</code> is set to NULL .

strict	No warning, no error; <code>i</code> is set to <code>NULL</code> .
<code>ERROR_FOR_DIVISION_BY_ZERO</code>	Warning, no error; <code>i</code> is set to <code>NULL</code> .
strict, <code>ERROR_FOR_DIVISION_BY_ZERO</code>	Error condition; no row is inserted.

For inserts of strings into numeric columns, conversion from string to number is handled as follows if the string has non-numeric contents:

- A string that does not begin with a number cannot be used as a number and produces an error in strict mode, or a warning otherwise. *This includes the empty string.*
- A string that begins with a number can be converted, but the trailing non-numeric portion is truncated. If the truncated portion contains anything other than spaces, this produces an error in strict mode, or a warning otherwise.

24.4. Rounding Behavior

This section discusses precision math rounding for the `ROUND()` function and for inserts into columns with exact-value types (`DECIMAL` and integer).

The `ROUND()` function rounds differently depending on whether its argument is exact or approximate:

- For exact-value numbers, `ROUND()` uses the “round half up” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` uses the “round to nearest even” rule: A value with any fractional part is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+-----+-----+
| ROUND(2.5) | ROUND(25E-1) |
+-----+-----+
| 3          | 2            |
+-----+-----+
```

For inserts into a `DECIMAL` or integer column, the target is an exact data type, so rounding uses “round half up,” regardless of whether the value to be inserted is exact or approximate:

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 2

mysql> SELECT d FROM t;
+-----+
| d    |
+-----+
| 3    |
| 3    |
+-----+
```

24.5. Precision Math Examples

This section provides some examples that show precision math query results in MySQL 5.1.

Example 1. Numbers are used with their exact value as given when possible:

```
mysql> SELECT .1 + .2 = .3;
+-----+
| .1 + .2 = .3 |
+-----+
```

```
+-----+
|          1 |
+-----+
```

For floating-point values, results are inexact:

```
mysql> SELECT .1E0 + .2E0 = .3E0;
+-----+
| .1E0 + .2E0 = .3E0 |
+-----+
|                      0 |
+-----+
```

Another way to see the difference in exact and approximate value handling is to add a small number to a sum many times. Consider the following stored procedure, which adds `.0001` to a variable 1,000 times.

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE d DECIMAL(10,4) DEFAULT 0;
  DECLARE f FLOAT DEFAULT 0;
  WHILE i < 10000 DO
    SET d = d + .0001;
    SET f = f + .0001E0;
    SET i = i + 1;
  END WHILE;
  SELECT d, f;
END;
```

The sum for both `d` and `f` logically should be 1, but that is true only for the decimal calculation. The floating-point calculation introduces small errors:

```
+-----+-----+
| d       | f       |
+-----+-----+
| 1.0000  | 0.999999999999991 |
+-----+-----+
```

Example 2. Multiplication is performed with the scale required by standard SQL. That is, for two numbers `X1` and `X2` that have scale `S1` and `S2`, the scale of the result is `S1 + S2`:

```
mysql> SELECT .01 * .01;
+-----+
| .01 * .01 |
+-----+
| 0.0001    |
+-----+
```

Example 3. Rounding behavior is well-defined:

Rounding behavior (for example, with the `ROUND()` function) is independent of the implementation of the underlying C library, which means that results are consistent from platform to platform.

Rounding for exact-value columns (`DECIMAL` and integer) and exact-valued numbers uses the “round half up” rule. Values with a fractional part of .5 or greater are rounded away from zero to the nearest integer, as shown here:

```
mysql> SELECT ROUND(2.5), ROUND(-2.5);
+-----+-----+
| ROUND(2.5) | ROUND(-2.5) |
+-----+-----+
| 3          | -3          |
+-----+-----+
```

However, rounding for floating-point values uses the C library, which on many systems uses the “round to nearest even” rule. Values with any fractional part on such systems are rounded to the nearest even integer:

```
mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
+-----+-----+
| ROUND(2.5E0) | ROUND(-2.5E0) |
+-----+-----+
| 2            | -2            |
+-----+-----+
```

Example 4. In strict mode, inserting a value that is too large results in overflow and causes an error, rather than truncation to a legal value.

When MySQL is not running in strict mode, truncation to a legal value occurs:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SELECT i FROM t;
+-----+
| i     |
+-----+
| 127   |
+-----+
1 row in set (0.00 sec)
```

However, an overflow condition occurs if strict mode is in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 128;
ERROR 1264 (22003): Out of range value adjusted for column 'i' at row 1

mysql> SELECT i FROM t;
Empty set (0.00 sec)
```

Example 5: In strict mode and with `ERROR_FOR_DIVISION_BY_ZERO` set, division by zero causes an error, and not a result of `NULL`.

In non-strict mode, division by zero has a result of `NULL`:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT i FROM t;
+-----+
| i     |
+-----+
| NULL  |
+-----+
1 row in set (0.03 sec)
```

However, division by zero is an error if the proper SQL modes are in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
ERROR 1365 (22012): Division by 0

mysql> SELECT i FROM t;
Empty set (0.01 sec)
```

Example 6. Prior to MySQL 5.0.3 (before precision math was introduced), exact-value and approximate-value literals both are converted to double-precision floating-point values:

```
mysql> SELECT VERSION();
+-----+
| VERSION() |
```

```
+-----+
| 4.1.18-log |
+-----+
1 row in set (0.01 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.07 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> DESCRIBE t;
+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+
| a      | double(3,1)   |      |     | 0.0     |       |
| b      | double        |      |     | 0       |       |
+-----+
2 rows in set (0.04 sec)
```

As of MySQL 5.0.3, the approximate-value literal still is converted to floating-point, but the exact-value literal is handled as **DECIMAL**:

```
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 5.1.6-alpha-log |
+-----+
1 row in set (0.11 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.01 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> DESCRIBE t;
+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+
| a      | decimal(2,1) unsigned | NO   |     | 0.0     |       |
| b      | double        | NO   |     | 0       |       |
+-----+
2 rows in set (0.01 sec)
```

Example 7. If the argument to an aggregate function is an exact numeric type, the result is also an exact numeric type, with a scale at least that of the argument.

Consider these statements:

```
mysql> CREATE TABLE t (i INT, d DECIMAL, f FLOAT);
mysql> INSERT INTO t VALUES(1,1,1);
mysql> CREATE TABLE y SELECT AVG(i), AVG(d), AVG(f) FROM t;
```

Result before MySQL 5.0.3 (prior to the introduction of precision math in MySQL):

```
mysql> DESCRIBE y;
+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+
| AVG(i) | double(17,4)  | YES  |     | NULL    |       |
| AVG(d) | double(17,4)  | YES  |     | NULL    |       |
| AVG(f) | double        | YES  |     | NULL    |       |
+-----+
```

The result is a double no matter the argument type.

Result as of MySQL 5.0.3:

```
mysql> DESCRIBE y;
+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+
| AVG(i) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(d) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(f) | double        | YES  |     | NULL    |       |
+-----+
```

The result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type.

Chapter 25. APIs and Libraries

This chapter describes the APIs available for MySQL, where to get them, and how to use them. The C API is the most extensively covered, because it was developed by the MySQL team, and is the basis for most of the other APIs. This chapter also covers the `libmysqld` library (the embedded server), as well as some programs that are useful for application developers.

25.1. libmysqld, the Embedded MySQL Server Library

25.1.1. Overview of the Embedded MySQL Server Library

The embedded MySQL server library makes it possible to run a full-featured MySQL server inside a client application. The main benefits are increased speed and more simple management for embedded applications.

The embedded server library is based on the client/server version of MySQL, which is written in C/C++. Consequently, the embedded server also is written in C/C++. There is no embedded server available in other languages.

The API is identical for the embedded MySQL version and the client/server version. To change an old threaded application to use the embedded library, you normally only have to add calls to the following functions:

Function	When to Call
<code>mysql_library_init()</code>	Should be called before any other MySQL function is called, preferably early in the <code>main()</code> function.
<code>mysql_library_end()</code>	Should be called before your program exits.
<code>mysql_thread_init()</code>	Should be called in each thread you create that accesses MySQL.
<code>mysql_thread_end()</code>	Should be called before calling <code>pthread_exit()</code>

Then you must link your code with `libmysqld.a` instead of `libmysqlclient.a`. To ensure binary compatibility between your application and the server library, be sure to compile your application against headers for the same series of MySQL that was used to compile the server library. For example, if `libmysqld` was compiled against MySQL 4.1 headers, do not compile your application against MySQL 5.1 headers, or vice versa.

The `mysql_library_xxx()` functions are also included in `libmysqlclient.a` to allow you to change between the embedded and the client/server version by just linking your application with the right library. See [Section 25.2.3.40](#), “`mysql_library_init()`”.

One difference between the embedded server and the standalone server is that for the embedded server, authentication for connections is disabled by default. To use authentication for the embedded server, specify the `--with-embedded-privilege-control` option when you invoke `configure` to configure your MySQL distribution.

25.1.2. Compiling Programs with `libmysqld`

To get a `libmysqld` library you should configure MySQL with the `--with-embedded-server` option. See [Section 2.9.2](#), “Typical configure Options”.

When you link your program with `libmysqld`, you must also include the system-specific `pthread` libraries and some libraries that the MySQL server uses. You can get the full list of libraries by executing `mysql_config --libmysqld-libs`.

The correct flags for compiling and linking a threaded program must be used, even if you do not directly call any thread functions in your code.

To compile a C program to include the necessary files to embed the MySQL server library into a compiled version of a program, use the GNU C compiler (`gcc`). The compiler will need to know where to find various files and need instructions on how to compile the program. The following example shows how a program could be compiled from the command line:

```
gcc mysql_test.c -o mysql_test -lz \
/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Immediately following the `gcc` command is the name of the uncompiled C program file. After it, the `-o` option is given to indicate that the file name that follows is the name that the compiler is to give to the output file, the compiled program. The next line of code tells the

compiler to obtain the location of the include files and libraries and other settings for the system on which it's compiled. Because of a problem with `mysql_config`, the option `-lz` (for compression) is added here. The `mysql_config` piece is contained in backticks, not single quotes.

25.1.3. Restrictions When Using the Embedded MySQL Server

The embedded server has the following limitations:

- No user-defined functions (UDFs).
- No stack trace on core dump.
- You cannot set this up as a master or a slave (no replication).
- Very large result sets may be unusable on low memory systems.
- You cannot connect to an embedded server from an outside process with sockets or TCP/IP. However, you can connect to an intermediate application, which in turn can connect to an embedded server on the behalf of a remote client or outside process.
- **InnoDB** is not reentrant in the embedded server and cannot be used for multiple connections, either successively or simultaneously.

Some of these limitations can be changed by editing the `mysql_embed.h` include file and recompiling MySQL.

25.1.4. Options with the Embedded Server

Any options that may be given with the `mysqld` server daemon, may be used with an embedded server library. Server options may be given in an array as an argument to the `mysql_library_init()`, which initializes the server. They also may be given in an option file like `my.cnf`. To specify an option file for a C program, use the `--defaults-file` option as one of the elements of the second argument of the `mysql_library_init()` function. See [Section 25.2.3.40](#), “`mysql_library_init()`”, for more information on the `mysql_library_init()` function.

Using option files can make it easier to switch between a client/server application and one where MySQL is embedded. Put common options under the `[server]` group. These are read by both MySQL versions. Client/server-specific options should go under the `[mysqld]` section. Put options specific to the embedded MySQL server library in the `[embedded]` section. Options specific to applications go under section labeled `[ApplicationName_SERVER]`. See [Section 4.3.2](#), “Using Option Files”.

25.1.5. Embedded Server Examples

These two example programs should work without any changes on a Linux or FreeBSD system. For other operating systems, minor changes are needed, mostly with file paths. These examples are designed to give enough details for you to understand the problem, without the clutter that is a necessary part of a real application. The first example is very straightforward. The second example is a little more advanced with some error checking. The first is followed by a command-line entry for compiling the program. The second is followed by a GNUmake file that may be used for compiling instead.

Example 1

`test1_libmysqld.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "mysql.h"

MYSQL *mysql;
MYSQL_RES *results;
MYSQL_ROW record;

static char *server_options[] = \
    { "mysql_test", "--defaults-file=my.cnf", NULL };
int num_elements = (sizeof(server_options) / sizeof(char *)) - 1;

static char *server_groups[] = { "libmysqld_server",
    "libmysqld_client", NULL };

int main(void)
{
    mysql_library_init(num_elements, server_options, server_groups);
    mysql = mysql_init(NULL);
    mysql_options(mysql, MYSQL_READ_DEFAULT_GROUP, "libmysqld_client");
```



```

mysql_options(mysql, MYSQL_OPT_USE_EMBEDDED_CONNECTION, NULL);
mysql_real_connect(mysql, NULL, NULL, NULL, "database1", 0, NULL, 0);
mysql_query(mysql, "SELECT column1, column2 FROM table1");
results = mysql_store_result(mysql);
while((record = mysql_fetch_row(results))) {
    printf("%s - %s \n", record[0], record[1]);
}
mysql_free_result(results);
mysql_close(mysql);
mysql_library_end();

return 0;
}

```

Here is the command line for compiling the above program:

```

gcc test1_libmysqld.c -o test1_libmysqld -lz \
/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`

```

Example 2

To try out the example, create an `test2_libmysqld` directory at the same level as the MySQL source directory. Save the `test2_libmysqld.c` source and the `GNUmakefile` in the directory, and run GNU `make` from inside the `test2_libmysqld` directory.

test2_libmysqld.c

```

/*
 * A simple example client, using the embedded MySQL server library
 */

#include <mysql.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

MYSQL *db_connect(const char *dbname);
void db_disconnect(MYSQL *db);
void db_do_query(MYSQL *db, const char *query);

const char *server_groups[] = {
    "test2_libmysqld_SERVER", "embedded", "server", NULL
};

int
main(int argc, char **argv)
{
    MYSQL *one, *two;

    /* mysql_library_init() must be called before any other mysql
     * functions.
     * You can use mysql_library_init(0, NULL, NULL), and it
     * initializes the server using groups = {
     *   "server", "embedded", NULL
     * }.
     * In your $HOME/.my.cnf file, you probably want to put:
     [test2_libmysqld_SERVER]
     language = /path/to/source/of/mysql/sql/share/english
     * You could, of course, modify argc and argv before passing
     * them to this function. Or you could create new ones in any
     * way you like. But all of the arguments in argv (except for
     * argv[0], which is the program name) should be valid options
     * for the MySQL server.
     * If you link this client against the normal mysqlclient
     * library, this function is just a stub that does nothing.
     */
    mysql_library_init(argc, argv, (char **)server_groups);

    one = db_connect("test");
    two = db_connect(NULL);

    db_do_query(one, "SHOW TABLE STATUS");
    db_do_query(two, "SHOW DATABASES");

```

```

mysql_close(two);
mysql_close(one);

/* This must be called after all other mysql functions */
mysql_library_end();

exit(EXIT_SUCCESS);
}

static void
die(MYSQL *db, char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    vfprintf(stderr, fmt, ap);
    va_end(ap);
    (void)putc('\n', stderr);
    if (db)
        db_disconnect(db);
    exit(EXIT_FAILURE);
}

MYSQL *
db_connect(const char *dbname)
{
    MYSQL *db = mysql_init(NULL);
    if (!db)
        die(db, "mysql_init failed: no memory");
    /*
     * Notice that the client and server use separate group names.
     * This is critical, because the server does not accept the
     * client's options, and vice versa.
     */
    mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test2_libmysqld_CLIENT");
    if (!mysql_real_connect(db, NULL, NULL, NULL, dbname, 0, NULL, 0))
        die(db, "mysql_real_connect failed: %s", mysql_error(db));

    return db;
}

void
db_disconnect(MYSQL *db)
{
    mysql_close(db);
}

void
db_do_query(MYSQL *db, const char *query)
{
    if (mysql_query(db, query) != 0)
        goto err;

    if (mysql_field_count(db) > 0)
    {
        MYSQL_RES *res;
        MYSQL_ROW row, end_row;
        int num_fields;

        if (!(res = mysql_store_result(db)))
            goto err;
        num_fields = mysql_num_fields(res);
        while ((row = mysql_fetch_row(res)))
        {
            (void)fputs(">> ", stdout);
            for (end_row = row + num_fields; row < end_row; ++row)
                (void)printf("%s\t", row ? (char*)*row : "NULL");
            (void)fputc('\n', stdout);
        }
        (void)fputc('\n', stdout);
        mysql_free_result(res);
    }
    else
        (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));

    return;
err:
    die(db, "db_do_query failed: %s [%s]", mysql_error(db), query);
}

```

GNUmakefile

```

# This assumes the MySQL software is installed in /usr/local/mysql
inc      := /usr/local/mysql/include/mysql
lib      := /usr/local/mysql/lib

# If you have not installed the MySQL software yet, try this instead
#inc     := $(HOME)/mysql-5.1/include

```

```
#lib      := $(HOME)/mysql-5.1/libmysqld

CC        := gcc
CPPFLAGS  := -I$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS    := -g -W -Wall
LDFLAGS    := -static
# You can change -lmysqld to -lmysqldclient to use the
# client/server library
LDLIBS     = -L$(lib) -lmysqld -lz -lm -lcrypt

ifneq (,$(shell grep FreeBSD /COPYRIGHT 2>/dev/null))
# FreeBSD
LDFLAGS += -pthread
else
# Assume Linux
LDLIBS += -lpthread
endif

# This works for simple one-file test programs
sources := $(wildcard *.c)
objects := $(patsubst %c,%o,$(sources))
targets := $(basename $(sources))

all: $(targets)

clean:
    rm -f $(targets) $(objects) *.core
```

25.1.6. Licensing the Embedded Server

We encourage everyone to promote free software by releasing code under the GPL or a compatible license. For those who are not able to do this, another option is to purchase a commercial license for the MySQL code from MySQL AB. For details, please see <http://www.mysql.com/company/legal/licensing/>.

25.2. MySQL C API

The C API code is distributed with MySQL. It is included in the `mysqlclient` library and allows C programs to access a database.

Many of the clients in the MySQL source distribution are written in C. If you are looking for examples that demonstrate how to use the C API, take a look at these clients. You can find these in the `clients` directory in the MySQL source distribution.

Most of the other client APIs (all except Connector/J and Connector/NET) use the `mysqlclient` library to communicate with the MySQL server. This means that, for example, you can take advantage of many of the same environment variables that are used by other client programs, because they are referenced from the library. See [Chapter 7, Client and Utility Programs](#), for a list of these variables.

The client has a maximum communication buffer size. The size of the buffer that is allocated initially (16KB) is automatically increased up to the maximum size (the maximum is 16MB). Because buffer sizes are increased only as demand warrants, simply increasing the default maximum limit does not in itself cause more resources to be used. This size check is mostly a check for erroneous statements and communication packets.

The communication buffer must be large enough to contain a single SQL statement (for client-to-server traffic) and one row of returned data (for server-to-client traffic). Each thread's communication buffer is dynamically enlarged to handle any query or row up to the maximum limit. For example, if you have `BLOB` values that contain up to 16MB of data, you must have a communication buffer limit of at least 16MB (in both server and client). The client's default maximum is 16MB, but the default maximum in the server is 1MB. You can increase this by changing the value of the `max_allowed_packet` parameter when the server is started. See [Section 6.5.2, "Tuning Server Parameters"](#).

The MySQL server shrinks each communication buffer to `net_buffer_length` bytes after each query. For clients, the size of the buffer associated with a connection is not decreased until the connection is closed, at which time client memory is reclaimed.

For programming with threads, see [Section 25.2.16, "How to Make a Threaded Client"](#). For creating a standalone application which includes the "server" and "client" in the same program (and does not communicate with an external MySQL server), see [Section 25.1, "libmysqld, the Embedded MySQL Server Library"](#).

MySQL Enterprise

MySQL Enterprise subscribers will find more information about using the C API in the Knowledge Base articles, [The C API](#). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

25.2.1. C API Data types

- `MYSQL`

This structure represents a handle to one database connection. It is used for almost all MySQL functions. You should not try to make a copy of a `MYSQL` structure. There is no guarantee that such a copy will be usable.

- `MYSQL_RES`

This structure represents the result of a query that returns rows (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). The information returned from a query is called the *result set* in the remainder of this section.

- `MYSQL_ROW`

This is a type-safe representation of one row of data. It is currently implemented as an array of counted byte strings. (You cannot treat these as null-terminated strings if field values may contain binary data, because such values may contain null bytes internally.) Rows are obtained by calling `mysql_fetch_row()`.

- `MYSQL_FIELD`

This structure contains information about a field, such as the field's name, type, and size. Its members are described in more detail here. You may obtain the `MYSQL_FIELD` structures for each field by calling `mysql_fetch_field()` repeatedly. Field values are not part of this structure; they are contained in a `MYSQL_ROW` structure.

- `MYSQL_FIELD_OFFSET`

This is a type-safe representation of an offset into a MySQL field list. (Used by `mysql_field_seek()`.) Offsets are field numbers within a row, beginning at zero.

- `my_ulonglong`

The type used for the number of rows and for `mysql_affected_rows()`, `mysql_num_rows()`, and `mysql_insert_id()`. This type provides a range of 0 to 1.84e19.

On some systems, attempting to print a value of type `my_ulonglong` does not work. To print such a value, convert it to `unsigned long` and use a `%lu` print format. Example:

```
printf ("Number of rows: %lu\n",
        (unsigned long) mysql_num_rows(result));
```

- `my_bool`

A boolean type, for values that are true (non-zero) or false (zero).

The `MYSQL_FIELD` structure contains the members listed here:

- `char * name`

The name of the field, as a null-terminated string. If the field was given an alias with an `AS` clause, the value of `name` is the alias.

- `char * org_name`

The name of the field, as a null-terminated string. Aliases are ignored.

- `char * table`

The name of the table containing this field, if it isn't a calculated field. For calculated fields, the `table` value is an empty string. If the column is selected from a view, `table` names the view. If the table or view was given an alias with an `AS` clause, the value of `table` is the alias.

- `char * org_table`

The name of the table, as a null-terminated string. Aliases are ignored. If the column is selected from a view, `org_table` names the underlying table.

- `char * db`

The name of the database that the field comes from, as a null-terminated string. If the field is a calculated field, `db` is an empty string.

- `char * catalog`

The catalog name. This value is always `"def"`.

- `char * def`

The default value of this field, as a null-terminated string. This is set only if you use `mysql_list_fields()`.

- `unsigned long length`

The width of the field. This corresponds to the display length, in bytes.

- `unsigned long max_length`

The maximum width of the field for the result set (the length in bytes of the longest field value for the rows actually in the result set). If you use `mysql_store_result()` or `mysql_list_fields()`, this contains the maximum length for the field. If you use `mysql_use_result()`, the value of this variable is zero.

The value of `max_length` is the length of the string representation of the values in the result set. For example, if you retrieve a `FLOAT` column and the “widest” value is `-12.345`, `max_length` is 7 (the length of `'-12.345'`).

If you are using prepared statements, `max_length` is not set by default because for the binary protocol the lengths of the values depend on the types of the values in the result set. (See [Section 25.2.5, “C API Prepared Statement Data types”](#).) If you want the `max_length` values anyway, enable the `STMT_ATTR_UPDATE_MAX_LENGTH` option with `mysql_stmt_attr_set()` and the lengths will be set when you call `mysql_stmt_store_result()`. (See [Section 25.2.7.3, “mysql_stmt_attr_set\(\)”](#), and [Section 25.2.7.27, “mysql_stmt_store_result\(\)”](#).)

- `unsigned int name_length`

The length of `name`.

- `unsigned int org_name_length`

The length of `org_name`.

- `unsigned int table_length`

The length of `table`.

- `unsigned int org_table_length`

The length of `org_table`.

- `unsigned int db_length`

The length of `db`.

- `unsigned int catalog_length`

The length of `catalog`.

- `unsigned int def_length`

The length of `def`.

- `unsigned int flags`

Different bit-flags for the field. The `flags` value may have zero or more of the following bits set:

Flag Value	Flag Description
<code>NOT_NULL_FLAG</code>	Field can't be <code>NULL</code>
<code>PRI_KEY_FLAG</code>	Field is part of a primary key

<code>UNIQUE_KEY_FLAG</code>	Field is part of a unique key
<code>MULTIPLE_KEY_FLAG</code>	Field is part of a non-unique key
<code>UNSIGNED_FLAG</code>	Field has the <code>UNSIGNED</code> attribute
<code>ZEROFILL_FLAG</code>	Field has the <code>ZEROFILL</code> attribute
<code>BINARY_FLAG</code>	Field has the <code>BINARY</code> attribute
<code>AUTO_INCREMENT_FLAG</code>	Field has the <code>AUTO_INCREMENT</code> attribute
<code>ENUM_FLAG</code>	Field is an <code>ENUM</code> (deprecated)
<code>SET_FLAG</code>	Field is a <code>SET</code> (deprecated)
<code>BLOB_FLAG</code>	Field is a <code>BLOB</code> or <code>TEXT</code> (deprecated)
<code>TIMESTAMP_FLAG</code>	Field is a <code>TIMESTAMP</code> (deprecated)
<code>NO_DEFAULT_VALUE_FLAG</code>	Field has no default value; see additional notes following table

Use of the `BLOB_FLAG`, `ENUM_FLAG`, `SET_FLAG`, and `TIMESTAMP_FLAG` flags is deprecated because they indicate the type of a field rather than an attribute of its type. It is preferable to test `field->type` against `MYSQL_TYPE_BLOB`, `MYSQL_TYPE_ENUM`, `MYSQL_TYPE_SET`, or `MYSQL_TYPE_TIMESTAMP` instead.

`NUM_FLAG` indicates that a column is numeric. This includes columns with a type of `MYSQL_TYPE_DECIMAL`, `MYSQL_TYPE_TINY`, `MYSQL_TYPE_SHORT`, `MYSQL_TYPE_LONG`, `MYSQL_TYPE_FLOAT`, `MYSQL_TYPE_DOUBLE`, `MYSQL_TYPE_NULL`, `MYSQL_TYPE_TIMESTAMP`, `MYSQL_TYPE_LONGLONG`, `MYSQL_TYPE_INT24`, and `MYSQL_TYPE_YEAR`.

`NO_DEFAULT_VALUE_FLAG` indicates that a column has no `DEFAULT` clause in its definition. This does not apply to `NULL` columns (because such columns have a default of `NULL`), or to `AUTO_INCREMENT` columns (which have an implied default value).

The following example illustrates a typical use of the `flags` value:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field can't be null\n");
```

You may use the following convenience macros to determine the boolean status of the `flags` value:

Flag Status	Description
<code>IS_NOT_NULL(flags)</code>	True if this field is defined as <code>NOT NULL</code>
<code>IS_PRI_KEY(flags)</code>	True if this field is a primary key
<code>IS_BLOB(flags)</code>	True if this field is a <code>BLOB</code> or <code>TEXT</code> (deprecated; test <code>field->type</code> instead)

- `unsigned int decimals`

The number of decimals for numeric fields.

- `unsigned int charsetnr`

The character set number for the field.

- `enum enum_field_types type`

The type of the field. The `type` value may be one of the `MYSQL_TYPE_` symbols shown in the following table.

Type Value	Type Description
<code>MYSQL_TYPE_TINY</code>	<code>TINYINT</code> field
<code>MYSQL_TYPE_SHORT</code>	<code>SMALLINT</code> field
<code>MYSQL_TYPE_LONG</code>	<code>INTEGER</code> field
<code>MYSQL_TYPE_INT24</code>	<code>MEDIUMINT</code> field
<code>MYSQL_TYPE_LONGLONG</code>	<code>BIGINT</code> field

<code>MYSQL_TYPE_DECIMAL</code>	<code>DECIMAL</code> or <code>NUMERIC</code> field
<code>MYSQL_TYPE_NEWDECIMAL</code>	Precision math <code>DECIMAL</code> or <code>NUMERIC</code>
<code>MYSQL_TYPE_FLOAT</code>	<code>FLOAT</code> field
<code>MYSQL_TYPE_DOUBLE</code>	<code>DOUBLE</code> or <code>REAL</code> field
<code>MYSQL_TYPE_BIT</code>	<code>BIT</code> field
<code>MYSQL_TYPE_TIMESTAMP</code>	<code>TIMESTAMP</code> field
<code>MYSQL_TYPE_DATE</code>	<code>DATE</code> field
<code>MYSQL_TYPE_TIME</code>	<code>TIME</code> field
<code>MYSQL_TYPE_DATETIME</code>	<code>DATETIME</code> field
<code>MYSQL_TYPE_YEAR</code>	<code>YEAR</code> field
<code>MYSQL_TYPE_STRING</code>	<code>CHAR</code> or <code>BINARY</code> field
<code>MYSQL_TYPE_VAR_STRING</code>	<code>VARCHAR</code> or <code>VARBINARY</code> field
<code>MYSQL_TYPE_BLOB</code>	<code>BLOB</code> or <code>TEXT</code> field (use <code>max_length</code> to determine the maximum length)
<code>MYSQL_TYPE_SET</code>	<code>SET</code> field
<code>MYSQL_TYPE_ENUM</code>	<code>ENUM</code> field
<code>MYSQL_TYPE_GEOMETRY</code>	Spatial field
<code>MYSQL_TYPE_NULL</code>	<code>NULL</code> -type field

You can use the `IS_NUM()` macro to test whether a field has a numeric type. Pass the `type` value to `IS_NUM()` and it evaluates to TRUE if the field is numeric:

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

To distinguish between binary and non-binary data for string data types, check whether the `charsetnr` value is 63. If so, the character set is `binary`, which indicates binary rather than non-binary data. This is how to distinguish between `BINARY` and `CHAR`, `VARBINARY` and `VARCHAR`, and `BLOB` and `TEXT`.

25.2.2. C API Function Overview

The functions available in the C API are summarized here and described in greater detail in a later section. See [Section 25.2.3, “C API Function Descriptions”](#).

Function	Description
<code>my_init()</code>	Initialize global variables, and thread handler in thread-safe programs.
<code>mysql_affected_rows()</code>	Returns the number of rows changed/deleted/inserted by the last <code>UPDATE</code> , <code>DELETE</code> , or <code>INSERT</code> query.
<code>mysql_autocommit()</code>	Toggles autocommit mode on/off.
<code>mysql_change_user()</code>	Changes user and database on an open connection.
<code>mysql_close()</code>	Closes a server connection.
<code>mysql_commit()</code>	Commits the transaction.
<code>mysql_connect()</code>	Connects to a MySQL server. This function is deprecated; use <code>mysql_real_connect()</code> instead.
<code>mysql_create_db()</code>	Creates a database. This function is deprecated; use the SQL statement <code>CREATE DATABASE</code> instead.
<code>mysql_data_seek()</code>	Seeks to an arbitrary row number in a query result set.
<code>mysql_debug()</code>	Does a <code>DEBUG_PUSH</code> with the given string.
<code>mysql_drop_db()</code>	Drops a database. This function is deprecated; use the SQL statement <code>DROP DATABASE</code> instead.

mysql_dump_debug_info()	Makes the server write debug information to the log.
mysql_eof()	Determines whether the last row of a result set has been read. This function is deprecated; mysql_errno() or mysql_error() may be used instead.
mysql_errno()	Returns the error number for the most recently invoked MySQL function.
mysql_error()	Returns the error message for the most recently invoked MySQL function.
mysql_escape_string()	Escapes special characters in a string for use in an SQL statement.
mysql_fetch_field()	Returns the type of the next table field.
mysql_fetch_field_direct()	Returns the type of a table field, given a field number.
mysql_fetch_fields()	Returns an array of all field structures.
mysql_fetch_lengths()	Returns the lengths of all columns in the current row.
mysql_fetch_row()	Fetches the next row from the result set.
mysql_field_seek()	Puts the column cursor on a specified column.
mysql_field_count()	Returns the number of result columns for the most recent statement.
mysql_field_tell()	Returns the position of the field cursor used for the last mysql_fetch_field() .
mysql_free_result()	Frees memory used by a result set.
mysql_get_client_info()	Returns client version information as a string.
mysql_get_client_version()	Returns client version information as an integer.
mysql_get_host_info()	Returns a string describing the connection.
mysql_get_server_version()	Returns version number of server as an integer.
mysql_get_proto_info()	Returns the protocol version used by the connection.
mysql_get_server_info()	Returns the server version number.
mysql_info()	Returns information about the most recently executed query.
mysql_init()	Gets or initializes a MYSQL structure.
mysql_insert_id()	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
mysql_kill()	Kills a given thread.
mysql_library_end()	Finalize the MySQL C API library.
mysql_library_init()	Initialize the MySQL C API library.
mysql_list_dbs()	Returns database names matching a simple regular expression.
mysql_list_fields()	Returns field names matching a simple regular expression.
mysql_list_processes()	Returns a list of the current server threads.
mysql_list_tables()	Returns table names matching a simple regular expression.
mysql_more_results()	Checks whether any more results exist.
mysql_next_result()	Returns/initiates the next result in multiple-statement executions.
mysql_num_fields()	Returns the number of columns in a result set.
mysql_num_rows()	Returns the number of rows in a result set.
mysql_options()	Sets connect options for mysql_connect() .
mysql_ping()	Checks whether the connection to the server is working, reconnecting as necessary.
mysql_query()	Executes an SQL query specified as a null-terminated string.
mysql_real_connect()	Connects to a MySQL server.
mysql_real_escape_string()	Escapes special characters in a string for use in an SQL statement, taking into account the current character set of the connection.
mysql_real_query()	Executes an SQL query specified as a counted string.
mysql_refresh()	Flush or reset tables and caches.
mysql_reload()	Tells the server to reload the grant tables.
mysql_rollback()	Rolls back the transaction.

mysql_row_seek()	Seeks to a row offset in a result set, using value returned from mysql_row_tell() .
mysql_row_tell()	Returns the row cursor position.
mysql_select_db()	Selects a database.
mysql_server_end()	Finalize the MySQL C API library.
mysql_server_init()	Initialize the MySQL C API library.
mysql_set_local_infile_default()	Set the LOAD DATA LOCAL INFILE handler callbacks to their default values.
mysql_set_local_infile_handler()	Install application-specific LOAD DATA LOCAL INFILE handler callbacks.
mysql_set_server_option()	Sets an option for the connection (like multi-statements).
mysql_sqlstate()	Returns the SQLSTATE error code for the last error.
mysql_shutdown()	Shuts down the database server.
mysql_stat()	Returns the server status as a string.
mysql_store_result()	Retrieves a complete result set to the client.
mysql_thread_end()	Finalize thread handler.
mysql_thread_id()	Returns the current thread ID.
mysql_thread_init()	Initialize thread handler.
mysql_thread_safe()	Returns 1 if the clients are compiled as thread-safe.
mysql_use_result()	Initiates a row-by-row result set retrieval.
mysql_warning_count()	Returns the warning count for the previous SQL statement.

Application programs should use this general outline for interacting with MySQL:

1. Initialize the MySQL library by calling [mysql_library_init\(\)](#). This function exists in both the [mysqlclient](#) C client library and the [mysqld](#) embedded server library, so it is used whether you build a regular client program by linking with the [-libmysqlclient](#) flag, or an embedded server application by linking with the [-libmysqld](#) flag.
2. Initialize a connection handler by calling [mysql_init\(\)](#) and connect to the server by calling [mysql_real_connect\(\)](#).
3. Issue SQL statements and process their results. (The following discussion provides more information about how to do this.)
4. Close the connection to the MySQL server by calling [mysql_close\(\)](#).
5. End use of the MySQL library by calling [mysql_library_end\(\)](#).

The purpose of calling [mysql_library_init\(\)](#) and [mysql_library_end\(\)](#) is to provide proper initialization and finalization of the MySQL library. For applications that are linked with the client library, they provide improved memory management. If you don't call [mysql_library_end\(\)](#), a block of memory remains allocated. (This does not increase the amount of memory used by the application, but some memory leak detectors will complain about it.) For applications that are linked with the embedded server, these calls start and stop the server.

In a non-multi-threaded environment, the call to [mysql_library_init\(\)](#) may be omitted, because [mysql_init\(\)](#) will invoke it automatically as necessary. However, [mysql_library_init\(\)](#) is not thread-safe in a multi-threaded environment, and thus neither is [mysql_init\(\)](#), which calls [mysql_library_init\(\)](#). You must either call [mysql_library_init\(\)](#) prior to spawning any threads, or else use a mutex to protect the call, whether you invoke [mysql_library_init\(\)](#) or indirectly via [mysql_init\(\)](#). This should be done prior to any other client library call.

To connect to the server, call [mysql_init\(\)](#) to initialize a connection handler, then call [mysql_real_connect\(\)](#) with that handler (along with other information such as the hostname, username, and password). Upon connection, [mysql_real_connect\(\)](#) sets the [reconnect](#) flag (part of the [MYSQL](#) structure) to a value of [1](#) in versions of the API older than 5.0.3, or [0](#) in newer versions. A value of [1](#) for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. You can use the [MYSQL_OPT_RECONNECT](#) option to [mysql_options\(\)](#) to control reconnection behavior. When you are done with the connection, call [mysql_close\(\)](#) to terminate it.

While a connection is active, the client may send SQL statements to the server using [mysql_query\(\)](#) or [mysql_real_query\(\)](#). The difference between the two is that [mysql_query\(\)](#) expects the query to be specified as a null-terminated string whereas

`mysql_real_query()` expects a counted string. If the string contains binary data (which may include null bytes), you must use `mysql_real_query()`.

For each non-**SELECT** query (for example, **INSERT**, **UPDATE**, **DELETE**), you can find out how many rows were changed (affected) by calling `mysql_affected_rows()`.

For **SELECT** queries, you retrieve the selected rows as a result set. (Note that some statements are **SELECT**-like in that they return rows. These include **SHOW**, **DESCRIBE**, and **EXPLAIN**. They should be treated the same way as **SELECT** statements.)

There are two ways for a client to process result sets. One way is to retrieve the entire result set all at once by calling `mysql_store_result()`. This function acquires from the server all the rows returned by the query and stores them in the client. The second way is for the client to initiate a row-by-row result set retrieval by calling `mysql_use_result()`. This function initializes the retrieval, but does not actually get any rows from the server.

In both cases, you access rows by calling `mysql_fetch_row()`. With `mysql_store_result()`, `mysql_fetch_row()` accesses rows that have previously been fetched from the server. With `mysql_use_result()`, `mysql_fetch_row()` actually retrieves the row from the server. Information about the size of the data in each row is available by calling `mysql_fetch_lengths()`.

After you are done with a result set, call `mysql_free_result()` to free the memory used for it.

The two retrieval mechanisms are complementary. Client programs should choose the approach that is most appropriate for their requirements. In practice, clients tend to use `mysql_store_result()` more commonly.

An advantage of `mysql_store_result()` is that because the rows have all been fetched to the client, you not only can access rows sequentially, you can move back and forth in the result set using `mysql_data_seek()` or `mysql_row_seek()` to change the current row position within the result set. You can also find out how many rows there are by calling `mysql_num_rows()`. On the other hand, the memory requirements for `mysql_store_result()` may be very high for large result sets and you are more likely to encounter out-of-memory conditions.

An advantage of `mysql_use_result()` is that the client requires less memory for the result set because it maintains only one row at a time (and because there is less allocation overhead, `mysql_use_result()` can be faster). Disadvantages are that you must process each row quickly to avoid tying up the server, you don't have random access to rows within the result set (you can only access rows sequentially), and you don't know how many rows are in the result set until you have retrieved them all. Furthermore, you **must** retrieve all the rows even if you determine in mid-retrieval that you've found the information you were looking for.

The API makes it possible for clients to respond appropriately to statements (retrieving rows only as necessary) without knowing whether the statement is a **SELECT**. You can do this by calling `mysql_store_result()` after each `mysql_query()` (or `mysql_real_query()`). If the result set call succeeds, the statement was a **SELECT** and you can read the rows. If the result set call fails, call `mysql_field_count()` to determine whether a result was actually to be expected. If `mysql_field_count()` returns zero, the statement returned no data (indicating that it was an **INSERT**, **UPDATE**, **DELETE**, and so forth), and was not expected to return rows. If `mysql_field_count()` is non-zero, the statement should have returned rows, but didn't. This indicates that the statement was a **SELECT** that failed. See the description for `mysql_field_count()` for an example of how this can be done.

Both `mysql_store_result()` and `mysql_use_result()` allow you to obtain information about the fields that make up the result set (the number of fields, their names and types, and so forth). You can access field information sequentially within the row by calling `mysql_fetch_field()` repeatedly, or by field number within the row by calling `mysql_fetch_field_direct()`. The current field cursor position may be changed by calling `mysql_field_seek()`. Setting the field cursor affects subsequent calls to `mysql_fetch_field()`. You can also get information for fields all at once by calling `mysql_fetch_fields()`.

For detecting and reporting errors, MySQL provides access to error information by means of the `mysql_errno()` and `mysql_error()` functions. These return the error code or error message for the most recently invoked function that can succeed or fail, allowing you to determine when an error occurred and what it was.

25.2.3. C API Function Descriptions

In the descriptions here, a parameter or return value of **NULL** means **NULL** in the sense of the C programming language, not a MySQL **NULL** value.

Functions that return a value generally return a pointer or an integer. Unless specified otherwise, functions returning a pointer return a non-**NULL** value to indicate success or a **NULL** value to indicate an error, and functions returning an integer return zero to indicate success or non-zero to indicate an error. Note that “non-zero” means just that. Unless the function description says otherwise, do not test against a value other than zero:

```
if (result)                /* correct */
    ... error ...
```

```
if (result < 0)           /* incorrect */
    ... error ...

if (result == -1)        /* incorrect */
    ... error ...
```

When a function returns an error, the **Errors** subsection of the function description lists the possible types of errors. You can find out which of these occurred by calling `mysql_errno()`. A string representation of the error may be obtained by calling `mysql_error()`.

MySQL Enterprise

MySQL Enterprise subscribers will find more information about the C API functions in the Knowledge Base articles, [The C API](#). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

25.2.3.1. `mysql_affected_rows()`

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Description

After executing a statement with `mysql_query()` or `mysql_real_query()`, returns the number of rows changed (for `UPDATE`), deleted (for `DELETE`, or inserted (for `INSERT`. For `SELECT` statements, `mysql_affected_rows()` works like `mysql_num_rows()`.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an `UPDATE` statement, no rows matched the `WHERE` clause in the query or that no query has yet been executed. -1 indicates that the query returned an error or that, for a `SELECT` query, `mysql_affected_rows()` was called prior to calling `mysql_store_result()`. Because `mysql_affected_rows()` returns an unsigned value, you can check for -1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent).

Errors

None.

Example

```
char *stmt = "UPDATE products SET cost=cost*1.25 WHERE group=10";
mysql_query(&mysql, stmt);
printf("%ld products updated",
       (long) mysql_affected_rows(&mysql));
```

For `UPDATE` statements, if you specify the `CLIENT_FOUND_ROWS` flag when connecting to `mysqld`, `mysql_affected_rows()` returns the number of rows matched by the `WHERE` clause. Otherwise, the default behavior is to return the number of rows actually changed.

Note that when you use a `REPLACE` command, `mysql_affected_rows()` returns 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

If you use `INSERT ... ON DUPLICATE KEY UPDATE` to insert a row, `mysql_affected_rows()` returns 1 if the row is inserted as a new row and 2 if an existing row is updated.

25.2.3.2. `mysql_autocommit()`

```
my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)
```

Description

Sets autocommit mode on if `mode` is 1, off if `mode` is 0.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

25.2.3.3. `mysql_change_user()`

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char *password, const char *db)
```

Description

Changes the user and causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

`mysql_change_user()` fails if the connected user cannot be authenticated or doesn't have permission to use the database. In this case, the user and database are not changed

The `db` parameter may be set to `NULL` if you don't want to have a default database.

This command resets the state as if one had done a new connect. (See [Section 25.2.13, “Controlling Automatic Reconnect Behavior”](#).) It always performs a `ROLLBACK` of any active transactions, closes and drops all temporary tables, and unlocks all locked tables. Session system variables are reset to the values of the corresponding global system variables. Prepared statements are released and `HANDLER` variables are closed. Locks acquired with `GET_LOCK()` are released. These effects occur even if the user didn't change.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

The same that you can get from `mysql_real_connect()`.

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR`
An unknown error occurred.
- `ER_UNKNOWN_COM_ERROR`
The MySQL server doesn't implement this command (probably an old server).
- `ER_ACCESS_DENIED_ERROR`
The user or password was wrong.
- `ER_BAD_DB_ERROR`
The database didn't exist.
- `ER_DBACCESS_DENIED_ERROR`
The user did not have access rights to the database.
- `ER_WRONG_DB_NAME`

The database name was too long.

Example

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
    fprintf(stderr, "Failed to change user.  Error: %s\n",
        mysql_error(&mysql));
}
```

25.2.3.4. `mysql_character_set_name()`

```
const char *mysql_character_set_name(MYSQL *mysql)
```

Description

Returns the default character set for the current connection.

Return Values

The default character set

Errors

None.

25.2.3.5. `mysql_close()`

```
void mysql_close(MYSQL *mysql)
```

Description

Closes a previously opened connection. `mysql_close()` also deallocates the connection handle pointed to by `mysql` if the handle was allocated automatically by `mysql_init()` or `mysql_connect()`.

Return Values

None.

Errors

None.

25.2.3.6. `mysql_commit()`

```
my_bool mysql_commit(MYSQL *mysql)
```

Description

Commits the current transaction.

The action of this function is subject to the value of the `completion_type` system variable. In particular, if the value of `completion_type` is 2, the server performs a release after terminating a transaction and closes the client connection. The client program should call `mysql_close()` to close the connection from the client side.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

25.2.3.7. `mysql_connect()`

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd)
```

Description

This function is deprecated. It is preferable to use `mysql_real_connect()` instead.

`mysql_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The meanings of the parameters are the same as for the corresponding parameters for `mysql_real_connect()` with the difference that the connection parameter may be `NULL`. In this case, the C API allocates memory for the connection structure automatically and frees it when you call `mysql_close()`. The disadvantage of this approach is that you can't retrieve an error message if the connection fails. (To get error information from `mysql_errno()` or `mysql_error()`, you must provide a valid `MYSQL` pointer.)

Return Values

Same as for `mysql_real_connect()`.

Errors

Same as for `mysql_real_connect()`.

25.2.3.8. `mysql_create_db()`

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

Description

Creates the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `CREATE DATABASE` statement instead.

Return Values

Zero if the database was created successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

Example

```
if(mysql_create_db(&mysql, "my_database"))
{
    fprintf(stderr, "Failed to create new database. Error: %s\n",
            mysql_error(&mysql));
}
```

25.2.3.9. `mysql_data_seek()`

```
void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a query result set. The `offset` value is a row number and should be in the range from `0` to `mysql_num_rows(result)-1`.

This function requires that the result set structure contains the entire result of the query, so `mysql_data_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

None.

Errors

None.

25.2.3.10. `mysql_debug()`

```
void mysql_debug(const char *debug)
```

Description

Does a `DEBUG_PUSH` with the given string. `mysql_debug()` uses the Fred Fish debug library. To use this function, you must compile the client library to support debugging. See [MySQL Internals: Porting](#).

Return Values

None.

Errors

None.

Example

The call shown here causes the client library to generate a trace file in `/tmp/client.trace` on the client machine:

```
mysql_debug("d:t:0,/tmp/client.trace");
```

25.2.3.11. `mysql_drop_db()`

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Description

Drops the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `DROP DATABASE` statement instead.

Return Values

Zero if the database was dropped successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR](#)

An unknown error occurred.

Example

```
if(mysql_drop_db(&mysql, "my_database"))
    fprintf(stderr, "Failed to drop the database: Error: %s\n",
            mysql_error(&mysql));
```

25.2.3.12. [mysql_dump_debug_info\(\)](#)

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Description

Instructs the server to write some debug information to the log. For this to work, the connected user must have the [SUPER](#) privilege.

Return Values

Zero if the command was successful. Non-zero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC](#)

Commands were executed in an improper order.

- [CR_SERVER_GONE_ERROR](#)

The MySQL server has gone away.

- [CR_SERVER_LOST](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR](#)

An unknown error occurred.

25.2.3.13. [mysql_eof\(\)](#)

```
my_bool mysql_eof(MYSQL_RES *result)
```

Description

This function is deprecated. [mysql_errno\(\)](#) or [mysql_error\(\)](#) may be used instead.

[mysql_eof\(\)](#) determines whether the last row of a result set has been read.

If you acquire a result set from a successful call to [mysql_store_result\(\)](#), the client receives the entire set in one operation. In this case, a [NULL](#) return from [mysql_fetch_row\(\)](#) always means the end of the result set has been reached and it is unnecessary to call [mysql_eof\(\)](#). When used with [mysql_store_result\(\)](#), [mysql_eof\(\)](#) always returns true.

On the other hand, if you use [mysql_use_result\(\)](#) to initiate a result set retrieval, the rows of the set are obtained from the server one by one as you call [mysql_fetch_row\(\)](#) repeatedly. Because an error may occur on the connection during this process, a [NULL](#) return value from [mysql_fetch_row\(\)](#) does not necessarily mean the end of the result set was reached normally. In this case, you can use [mysql_eof\(\)](#) to determine what happened. [mysql_eof\(\)](#) returns a non-zero value if the end of the result set was reached and zero if an error occurred.

Historically, `mysql_eof()` predates the standard MySQL error functions `mysql_errno()` and `mysql_error()`. Because those error functions provide the same information, their use is preferred over `mysql_eof()`, which is deprecated. (In fact, they provide more information, because `mysql_eof()` returns only a boolean value whereas the error functions indicate a reason for the error when one occurs.)

Return Values

Zero if no error occurred. Non-zero if the end of the result set has been reached.

Errors

None.

Example

The following example shows how you might use `mysql_eof()`:

```
mysql_query(&mysql, "SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result)) // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

However, you can achieve the same effect with the standard MySQL error functions:

```
mysql_query(&mysql, "SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql)) // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

25.2.3.14. `mysql_errno()`

```
unsigned int mysql_errno(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_errno()` returns the error code for the most recently invoked API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at [Appendix B, Errors, Error Codes, and Common Problems](#).

Note that some functions like `mysql_fetch_row()` don't set `mysql_errno()` if they succeed.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_errno()` if they succeed.

MySQL-specific error numbers returned by `mysql_errno()` differ from SQLSTATE values returned by `mysql_sqlstate()`. For example, the `mysql` client program displays errors using the following format, where `1146` is the `mysql_errno()` value and `'42S02'` is the corresponding `mysql_sqlstate()` value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Return Values

An error code value for the last `mysql_xxx()` call, if it failed. zero means no error occurred.

Errors

None.

25.2.3.15. `mysql_error()`

```
const char *mysql_error(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_error()` returns a null-terminated string containing the error message for the most recently invoked API function that failed. If a function didn't fail, the return value of `mysql_error()` may be the previous error or an empty string to indicate no error.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_error()` if they succeed.

For functions that reset `mysql_error()`, the following two tests are equivalent:

```
if(*mysql_error(&mysql))
{
    // an error occurred
}

if(mysql_error(&mysql)[0])
{
    // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently, you can choose error messages in several different languages. See [Section 5.10.2, “Setting the Error Message Language”](#).

Return Values

A null-terminated character string that describes the error. An empty string if no error occurred.

Errors

None.

25.2.3.16. `mysql_escape_string()`

You should use `mysql_real_escape_string()` instead!

This function is identical to `mysql_real_escape_string()` except that `mysql_real_escape_string()` takes a connection handler as its first argument and escapes the string according to the current character set. `mysql_escape_string()` does not take a connection argument and does not respect the current character set.

25.2.3.17. `mysql_fetch_field()`

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

Description

Returns the definition of one column of a result set as a `MYSQL_FIELD` structure. Call this function repeatedly to retrieve information about all columns in the result set. `mysql_fetch_field()` returns `NULL` when no more fields are left.

`mysql_fetch_field()` is reset to return information about the first field each time you execute a new `SELECT` query. The field returned by `mysql_fetch_field()` is also affected by calls to `mysql_field_seek()`.

If you've called `mysql_query()` to perform a `SELECT` on a table but have not called `mysql_store_result()`, MySQL returns the default blob length (8KB) if you call `mysql_fetch_field()` to ask for the length of a `BLOB` field. (The 8KB size is chosen because MySQL doesn't know the maximum length for the `BLOB`. This should be made configurable sometime.) Once you've retrieved the result set, `field->max_length` contains the length of the largest value for this column in the specific query.

Return Values

The `MYSQL_FIELD` structure for the current column. `NULL` if no columns are left.

Errors

None.

Example

```
MYSQL_FIELD *field;
while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

25.2.3.18. `mysql_fetch_field_direct()`

`MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int fieldnr)`

Description

Given a field number `fieldnr` for a column within a result set, returns that column's field definition as a `MYSQL_FIELD` structure. You may use this function to retrieve the definition for an arbitrary column. The value of `fieldnr` should be in the range from 0 to `mysql_num_fields(result)-1`.

Return Values

The `MYSQL_FIELD` structure for the specified column.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}
```

25.2.3.19. `mysql_fetch_fields()`

`MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)`

Description

Returns an array of all `MYSQL_FIELD` structures for a result set. Each structure provides the field definition for one column of the result set.

Return Values

An array of `MYSQL_FIELD` structures for all columns of a result set.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
    printf("Field %u is %s\n", i, fields[i].name);
}
```

25.2.3.20. `mysql_fetch_lengths()`

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Description

Returns the lengths of the columns of the current row within a result set. If you plan to copy field values, this length information is also useful for optimization, because you can avoid calling `strlen()`. In addition, if the result set contains binary data, you **must** use this function to determine the size of the data, because `strlen()` returns incorrect results for any field containing null characters.

The length for empty columns and for columns containing `NULL` values is zero. To see how to distinguish these two cases, see the description for `mysql_fetch_row()`.

Return Values

An array of unsigned long integers representing the size of each column (not including any terminating null characters). `NULL` if an error occurred.

Errors

`mysql_fetch_lengths()` is valid only for the current row of the result set. It returns `NULL` if you call it before calling `mysql_fetch_row()` or after retrieving all rows in the result.

Example

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("Column %u is %lu bytes in length.\n",
               i, lengths[i]);
    }
}
```

25.2.3.21. `mysql_fetch_row()`

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

Description

Retrieves the next row of a result set. When used after `mysql_store_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve. When used after `mysql_use_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve or if an error occurred.

The number of values in the row is given by `mysql_num_fields(result)`. If `row` holds the return value from a call to `mysql_fetch_row()`, pointers to the values are accessed as `row[0]` to `row[mysql_num_fields(result)-1]`. `NULL` values in the row are indicated by `NULL` pointers.

The lengths of the field values in the row may be obtained by calling `mysql_fetch_lengths()`. Empty fields and fields containing `NULL` both have length 0; you can distinguish these by checking the pointer for the field value. If the pointer is `NULL`, the field is `NULL`; otherwise, the field is empty.

Return Values

A `MYSQL_ROW` structure for the next row. `NULL` if there are no more rows to retrieve or if an error occurred.

Errors

Note that error is not reset between calls to `mysql_fetch_row()`

- [CR_SERVER_LOST](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR](#)

An unknown error occurred.

Example

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
    unsigned long *lengths;
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("[%s] ", (int) lengths[i],
                row[i] ? row[i] : "NULL");
    }
    printf("\n");
}
```

25.2.3.22. [mysql_field_count\(\)](#)

`unsigned int mysql_field_count(MYSQL *mysql)`

Description

Returns the number of columns for the most recent query on the connection.

The normal use of this function is when [mysql_store_result\(\)](#) returned `NULL` (and thus you have no result set pointer). In this case, you can call [mysql_field_count\(\)](#) to determine whether [mysql_store_result\(\)](#) should have produced a non-empty result. This allows the client program to take proper action without knowing whether the query was a [SELECT](#) (or [SELECT](#)-like) statement. The example shown here illustrates how this may be done.

See [Section 25.2.14.1, “Why mysql_store_result\(\) Sometimes Returns NULL After mysql_query\(\) Returns Success”](#).

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql, query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
```

```
    }
    else // mysql_store_result() should have returned data
    {
        fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
    }
}
```

An alternative is to replace the `mysql_field_count(&mysql)` call with `mysql_errno(&mysql)`. In this case, you are checking directly for an error from `mysql_store_result()` rather than inferring from the value of `mysql_field_count()` whether the statement was a `SELECT`.

25.2.3.23. `mysql_field_seek()`

`MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET offset)`

Description

Sets the field cursor to the given offset. The next call to `mysql_fetch_field()` retrieves the field definition of the column associated with that offset.

To seek to the beginning of a row, pass an `offset` value of zero.

Return Values

The previous value of the field cursor.

Errors

None.

25.2.3.24. `mysql_field_tell()`

`MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)`

Description

Returns the position of the field cursor used for the last `mysql_fetch_field()`. This value can be used as an argument to `mysql_field_seek()`.

Return Values

The current offset of the field cursor.

Errors

None.

25.2.3.25. `mysql_free_result()`

`void mysql_free_result(MYSQL_RES *result)`

Description

Frees the memory allocated for a result set by `mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()`, and so forth. When you are done with a result set, you must free the memory it uses by calling `mysql_free_result()`.

Do not attempt to access a result set after freeing it.

Return Values

None.

Errors

None.

25.2.3.26. `mysql_get_character_set_info()`

```
void mysql_get_character_set_info(MYSQL *mysql, MY_CHARSET_INFO *cs)
```

Description

This function provides information about the default client character set. The default character set may be changed with the `mysql_set_character_set()` function.

Example

```
if (!mysql_set_character_set(&mysql, "utf8"))
{
    MY_CHARSET_INFO cs;
    mysql_get_character_set_info(&mysql, &cs);
    printf("character set information:\n");
    printf("character set name: %s\n", cs.name);
    printf("collation name: %s\n", cs.csname);
    printf("comment: %s\n", cs.comment);
    printf("directory: %s\n", cs.dir);
    printf("multi byte character min. length: %d\n", cs.mbminlen);
    printf("multi byte character max. length: %d\n", cs.mbmaxlen);
}
```

25.2.3.27. `mysql_get_client_info()`

```
const char *mysql_get_client_info(void)
```

Description

Returns a string that represents the client library version.

Return Values

A character string that represents the MySQL client library version.

Errors

None.

25.2.3.28. `mysql_get_client_version()`

```
unsigned long mysql_get_client_version(void)
```

Description

Returns an integer that represents the client library version. The value has the format `XYZZ` where `X` is the major version, `YY` is the release level, and `ZZ` is the version number within the release level. For example, a value of `40102` represents a client library version of `4.1.2`.

Return Values

An integer that represents the MySQL client library version.

Errors

None.

25.2.3.29. `mysql_get_host_info()`

```
const char *mysql_get_host_info(MYSQL *mysql)
```

Description

Returns a string describing the type of connection in use, including the server hostname.

Return Values

A character string representing the server hostname and the connection type.

Errors

None.

25.2.3.30. `mysql_get_proto_info()`

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

Description

Returns the protocol version used by current connection.

Return Values

An unsigned integer representing the protocol version used by the current connection.

Errors

None.

25.2.3.31. `mysql_get_server_info()`

```
const char *mysql_get_server_info(MYSQL *mysql)
```

Description

Returns a string that represents the server version number.

Return Values

A character string that represents the server version number.

Errors

None.

25.2.3.32. `mysql_get_server_version()`

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Description

Returns the version number of the server as an integer.

Return Values

A number that represents the MySQL server version in this format:

```
major_version*10000 + minor_version *100 + sub_version
```

For example, 5.1.5 is returned as 50105.

This function is useful in client programs for quickly determining whether some version-specific server capability exists.

Errors

None.

25.2.3.33. `mysql_get_ssl_cipher()`

```
const char *mysql_get_ssl_cipher(MYSQL *mysql)
```

Description

`mysql_get_ssl_cipher()` returns the SSL cipher used for the given connection to the server. `mysql` is the connection handler returned from `mysql_init()`.

This function was added in MySQL 5.1.9.

Return Values

A string naming the SSL cipher used for the connection, or `NULL` if no cipher is being used.

25.2.3.34. `mysql_hex_string()`

`unsigned long mysql_hex_string(char *to, const char *from, unsigned long length)`

Description

This function is used to create a legal SQL string that you can use in an SQL statement. See [Section 8.1.1, “Strings”](#).

The string in `from` is encoded to hexadecimal format, with each character encoded as two hexadecimal digits. The result is placed in `to` and a terminating null byte is appended.

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. When `mysql_hex_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

The return value can be placed into an SQL statement using either `0xvalue` or `X'value` format. However, the return value does not include the `0x` or `X'...`. The caller must supply whichever of those is desired.

Example

```
char query[1000], *end;

end = strmov(query, "INSERT INTO test_table values(");
end = strmov(end, "0x");
end += mysql_hex_string(end, "What's this", 11);
end = strmov(end, "0x");
end += mysql_hex_string(end, "binary data: \0\r\n", 16);
*end++ = ')';

if (mysql_real_query(&mysql, query, (unsigned int) (end - query)))
{
    fprintf(stderr, "Failed to insert row, Error: %s\n",
        mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

25.2.3.35. `mysql_info()`

`const char *mysql_info(MYSQL *mysql)`

Description

Retrieves a string providing information about the most recently executed statement, but only for the statements listed here. For other statements, `mysql_info()` returns `NULL`. The format of the string varies depending on the type of statement, as described here. The numbers are illustrative only; the string contains values appropriate for the statement.

- `INSERT INTO ... SELECT ...`

String format: `Records: 100 Duplicates: 0 Warnings: 0`

- `INSERT INTO ... VALUES (...),(...),(...)...`

String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `LOAD DATA INFILE ...`

String format: `Records: 1 Deleted: 0 Skipped: 0 Warnings: 0`

- `ALTER TABLE`

String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `UPDATE`

String format: `Rows matched: 40 Changed: 40 Warnings: 0`

Note that `mysql_info()` returns a non-`NULL` value for `INSERT ... VALUES` only for the multiple-row form of the statement (that is, only if multiple value lists are specified).

Return Values

A character string representing additional information about the most recently executed statement. `NULL` if no information is available for the statement.

Errors

None.

25.2.3.36. `mysql_init()`

```
MYSQL *mysql_init(MYSQL *mysql)
```

Description

Allocates or initializes a `MYSQL` object suitable for `mysql_real_connect()`. If `mysql` is a `NULL` pointer, the function allocates, initializes, and returns a new object. Otherwise, the object is initialized and the address of the object is returned. If `mysql_init()` allocates a new object, it is freed when `mysql_close()` is called to close the connection.

Return Values

An initialized `MYSQL*` handle. `NULL` if there was insufficient memory to allocate a new object.

Errors

In case of insufficient memory, `NULL` is returned.

25.2.3.37. `mysql_insert_id()`

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the previous `INSERT` statement. Use this function after you have performed an `INSERT` statement into a table that contains an `AUTO_INCREMENT` field.

The return value of `mysql_insert_id()` is always zero unless explicitly updated under one of the following conditions:

- `INSERT` statements that store a value into an `AUTO_INCREMENT` column. This is true whether the value is automatically generated by storing the special values `NULL` or `0` into the column, or is an explicit non-special value.
- In the case of a multiple-row `INSERT` statement, the return value of `mysql_insert_id()` depends on the MySQL server version.

In MySQL 5.1.12 and later, `mysql_insert_id()` returns the **first** automatically generated `AUTO_INCREMENT` value that was *success-*

fully inserted. In MySQL 5.1.11 and earlier, `mysql_insert_id()` returns the **first** automatically generated `AUTO_INCREMENT` value, regardless of whether the insertion of that value was successful.

If no rows are successfully inserted, then `mysql_insert_id()` returns 0.

- Starting in MySQL 5.1.12, if an `INSERT ... SELECT` statement is executed, and no automatically generated value is successfully inserted, then `mysql_insert_id()` returns the id of the last inserted row.
- Starting in MySQL 5.1.12, if an `INSERT ... SELECT` statement uses `LAST_INSERT_ID(expr)`, then `mysql_insert_id()` returns `expr`.
- `INSERT` statements that generate an `AUTO_INCREMENT` value by inserting `LAST_INSERT_ID(expr)` into any column.
- `INSERT` statements that generate an `AUTO_INCREMENT` value by updating any column to `LAST_INSERT_ID(expr)`.
- The value of `mysql_insert_id()` is not affected by statements such as `SELECT` that return a result set.
- If the previous statement returned an error, the value of `mysql_insert_id()` is undefined.

For 5.1.12 and later, the return value of `mysql_insert_id()` can be simplified to the following sequence:

1. If there is an auto increment column, and an automatically generated value was successfully inserted, return the value of the first such value.
2. If there was a `LAST_INSERT_ID(EXPR)` in the statement, return `EXPR`, even if there was an auto increment column in the affected table.
3. The return value varies depending on the statement used. When called after an `INSERT INTO` statement:
 - If there is an auto increment column in the table, and there were some explicit values for this column that were successfully inserted into the table, then return the last of the explicit values.

When called after an `INSERT ... ON DUPLICATE KEY` statement:

- If there is an auto increment column in the table and there were some explicit successfully inserted values, or some updated rows, then return the last of the inserted or updated values.

Note

`mysql_insert_id()` returns 0 if the previous statement does not use an `AUTO_INCREMENT` value. If you need to save the value for later, be sure to call `mysql_insert_id()` immediately after the statement that generates the value.

The value of `mysql_insert_id()` is affected only by statements issued within the current client connection. It is not affected by statements issued by other clients.

See [Section 11.11.3, “Information Functions”](#).

Also note that the value of the SQL `LAST_INSERT_ID()` function will contain the value of the first automatically generated value that was successfully inserted (starting from 5.1.12) or the first automatically generated value if any rows were successfully inserted (before 5.1.12). Another difference is that `LAST_INSERT_ID()` is not updated if you set an `AUTO_INCREMENT` column to a specific non-special value.

The reason for the difference between `LAST_INSERT_ID()` and `mysql_insert_id()` is that `LAST_INSERT_ID()` is made easy to use in scripts while `mysql_insert_id()` tries to provide a little more exact information of what happens to the `AUTO_INCREMENT` column.

Return Values

Described in the preceding discussion.

Errors

None.

25.2.3.38. `mysql_kill()`

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Description

Asks the server to kill the thread specified by `pid`.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

25.2.3.39. `mysql_library_end()`

```
void mysql_library_end(void)
```

Description

This function finalizes the MySQL library. You should call it when you are done using the library (for example, after disconnecting from the server). The action taken by the call depends on whether your application is linked to the MySQL client library or the MySQL embedded server library. For a client program linked against the `libmysqlclient` library by using the `-lmysqlclient` flag, `mysql_library_end()` performs some memory management to clean up. For an embedded server application linked against the `libmysqld` library by using the `-lmysqld` flag, `mysql_library_end()` shuts down the embedded server and then cleans up.

See [Section 25.2.2, “C API Function Overview”](#), and [Section 25.2.3.40, “`mysql\_library\_init\(\)`”](#), for usage information.

25.2.3.40. `mysql_library_init()`

```
int mysql_library_init(int argc, char **argv, char **groups)
```

Description

This function should be called to initialize the MySQL library before you call any other MySQL function. If your application uses the embedded server, this call starts the server and initializes any subsystems (`mysys`, `InnoDB`, and so forth) that the server uses.

In a non-multi-threaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multi-threaded environment, and thus neither is `mysql_init()`, which calls `mysql_library_init()`. You must either call `mysql_library_init()` prior to spawning any threads, or else use a mutex to protect the call, whether you invoke `mysql_library_init()` or indirectly via `mysql_init()`. This should be done prior to any other client library call.

After your application is done using the MySQL library, call `mysql_library_end()` to clean up. See [Section 25.2.3.39, “`mysql\_library\_end\(\)`”](#).

The `argc` and `argv` arguments are analogous to the arguments to `main()`. The first element of `argv` is ignored (it typically contains

the program name). For convenience, `argc` may be 0 (zero) if there are no command-line arguments for the server. `mysql_library_init()` makes a copy of the arguments so it's safe to destroy `argv` or `groups` after the call.

If you want to connect to an external server without starting the embedded server, you have to specify a negative value for `argc`.

The `groups` argument should be an array of strings that indicate the groups in option files from which options should be read. See [Section 4.3.2, “Using Option Files”](#). The final entry in the array should be `NULL`. For convenience, if the `groups` argument itself is `NULL`, the `[server]` and `[embedded]` groups are used by default.

See [Section 25.2.2, “C API Function Overview”](#), for additional usage information.

Example

```
#include <mysql.h>
#include <stdlib.h>

static char *server_args[] = {
    "this_program",          /* this string is not used */
    "--datadir=",
    "--key_buffer_size=32M"
};
static char *server_groups[] = {
    "embedded",
    "server",
    "this_program_SERVER",
    (char *)NULL
};

int main(void) {
    if (mysql_library_init(sizeof(server_args) / sizeof(char *),
                          server_args, server_groups)) {
        fprintf(stderr, "could not initialize MySQL library\n");
        exit(1);
    }

    /* Use any MySQL API functions here */

    mysql_library_end();

    return EXIT_SUCCESS;
}
```

Return Values

Zero if successful. Non-zero if an error occurred.

25.2.3.41. `mysql_list_dbs()`

`MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)`

Description

Returns a result set consisting of database names on the server that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters ‘%’ or ‘_’, or may be a `NULL` pointer to match all databases. Calling `mysql_list_dbs()` is similar to executing the query `SHOW databases [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_OUT_OF_MEMORY`
Out of memory.
- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- [CR_SERVER_LOST](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR](#)

An unknown error occurred.

25.2.3.42. [mysql_list_fields\(\)](#)

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)
```

Description

Returns a result set consisting of field names in the given table that match the simple regular expression specified by the *wild* parameter. *wild* may contain the wildcard characters ‘%’ or ‘_’, or may be a `NULL` pointer to match all fields. Calling [mysql_list_fields\(\)](#) is similar to executing the query `SHOW COLUMNS FROM tbl_name [LIKE wild]`.

Note that it's recommended that you use `SHOW COLUMNS FROM tbl_name` instead of [mysql_list_fields\(\)](#).

You must free the result set with [mysql_free_result\(\)](#).

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC](#)

Commands were executed in an improper order.

- [CR_SERVER_GONE_ERROR](#)

The MySQL server has gone away.

- [CR_SERVER_LOST](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR](#)

An unknown error occurred.

25.2.3.43. [mysql_list_processes\(\)](#)

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

Description

Returns a result set describing the current server threads. This is the same kind of information as that reported by `mysqladmin processlist` or a `SHOW PROCESSLIST` query.

You must free the result set with [mysql_free_result\(\)](#).

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR](#)
The MySQL server has gone away.
- [CR_SERVER_LOST](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR](#)
An unknown error occurred.

25.2.3.44. [mysql_list_tables\(\)](#)

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of table names in the current database that match the simple regular expression specified by the [wild](#) parameter. [wild](#) may contain the wildcard characters ‘%’ or ‘_’, or may be a [NULL](#) pointer to match all tables. Calling [mysql_list_tables\(\)](#) is similar to executing the query `SHOW tables [LIKE wild]`.

You must free the result set with [mysql_free_result\(\)](#).

Return Values

A [MYSQL_RES](#) result set for success. [NULL](#) if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR](#)
The MySQL server has gone away.
- [CR_SERVER_LOST](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR](#)
An unknown error occurred.

25.2.3.45. [mysql_more_results\(\)](#)

```
my_bool mysql_more_results(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string, or when you execute [CALL](#) statements, which can return multiple result sets.

[mysql_more_results\(\)](#) true if more results exist from the currently executed statement, in which case the application must call [mysql_next_result\(\)](#) to fetch the results.

Return Values

`TRUE` (1) if more results exist. `FALSE` (0) if no more results exist.

In most cases, you can call `mysql_next_result()` instead to test whether more results exist and initiate retrieval if so.

See [Section 25.2.9, “C API Handling of Multiple Statement Execution”](#), and [Section 25.2.3.46, “mysql_next_result\(\)”](#).

Errors

None.

25.2.3.46. `mysql_next_result()`

```
int mysql_next_result(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string, or when you execute `CALL` statements, which can return multiple result sets.

If more statement results exist, `mysql_next_result()` reads the next statement result and returns the status back to the application.

Before calling `mysql_next_result()`, you must call `mysql_free_result()` for the preceding statement if it is a query that returned a result set.

After calling `mysql_next_result()` the state of the connection is as if you had called `mysql_real_query()` or `mysql_query()` for the next statement. This means that you can call `mysql_store_result()`, `mysql_warning_count()`, `mysql_affected_rows()`, and so forth.

If `mysql_next_result()` returns an error, no other statements are executed and there are no more results to fetch.

If your program executes stored procedures with the `CALL` SQL statement, you *must* set the `CLIENT_MULTI_RESULTS` flag explicitly, or implicitly by setting `CLIENT_MULTI_STATEMENTS` when you call `mysql_real_connect()`. This is because each `CALL` returns a result to indicate the call status, in addition to any results sets that might be returned by statements executed within the procedure. In addition, because `CALL` can return multiple results, you should process those results using a loop that calls `mysql_next_result()` to determine whether there are more results.

For an example that shows how to use `mysql_next_result()`, see [Section 25.2.9, “C API Handling of Multiple Statement Execution”](#).

Return Values

Return Value	Description
0	Successful and there are more results
-1	Successful and there are no more results
>0	An error occurred

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order. For example if you didn't call `mysql_use_result()` for a previous result set.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

25.2.3.47. `mysql_num_fields()`

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

To pass a `MYSQL *` argument instead, use `unsigned int mysql_field_count(MYSQL *mysql)`.

Description

Returns the number of columns in a result set.

Note that you can get the number of columns either from a pointer to a result set or to a connection handle. You would use the connection handle if `mysql_store_result()` or `mysql_use_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a non-empty result. This allows the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See [Section 25.2.14.1, “Why `mysql\_store\_result\(\)` Sometimes Returns `NULL` After `mysql\_query\(\)` Returns Success](#)”.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql, query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(&mysql))
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
```

An alternative (if you know that your query should have returned a result set) is to replace the `mysql_errno(&mysql)` call with a check whether `mysql_field_count(&mysql)` returns 0. This happens only if something went wrong.

25.2.3.48. `mysql_num_rows()`

```
my_ulonglong mysql_num_rows(MYSQL_RES *result)
```

Description

Returns the number of rows in the result set.

The use of `mysql_num_rows()` depends on whether you use `mysql_store_result()` or `mysql_use_result()` to return the result set. If you use `mysql_store_result()`, `mysql_num_rows()` may be called immediately. If you use `mysql_use_result()`, `mysql_num_rows()` does not return the correct value until all the rows in the result set have been retrieved.

`mysql_num_rows()` is intended for use with statements that return a result set, such as `SELECT`. For statements such as `INSERT`, `UPDATE`, or `DELETE`, the number of affected rows can be obtained with `mysql_affected_rows()`.

Return Values

The number of rows in the result set.

Errors

None.

25.2.3.49. `mysql_options()`

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const void *arg)
```

Description

Can be used to set extra connect options and affect behavior for a connection. This function may be called multiple times to set several options.

`mysql_options()` should be called after `mysql_init()` and before `mysql_connect()` or `mysql_real_connect()`.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, `arg` should point to the value of the integer.

The following list describes the possible options, their effect, and how `arg` is used for each option. Several of the options apply only when the application is linked against the `libmysql` embedded server library and are unused for applications linked against the `libmysql` client library. For option descriptions that indicate `arg` is unused, its value is irrelevant; it is conventional to pass 0.

- `MYSQL_INIT_COMMAND` (argument type: `char *`)
Statement to execute when connecting to the MySQL server. Automatically re-executed if reconnection occurs.
- `MYSQL_OPT_COMPRESS` (argument: not used)
Use the compressed client/server protocol.
- `MYSQL_OPT_CONNECT_TIMEOUT` (argument type: `unsigned int *`)
Connect timeout in seconds.
- `MYSQL_OPT_GUESS_CONNECTION` (argument: not used)
For an application linked against the `libmysql` embedded server library, this allows the library to guess whether to use the embedded server or a remote server. “Guess” means that if the hostname is set and is not `localhost`, it uses a remote server. This behavior is the default. `MYSQL_OPT_USE_EMBEDDED_CONNECTION` and `MYSQL_OPT_USE_REMOTE_CONNECTION` can be used to override it. This option is ignored for applications linked against the `libmysqlclient` client library.
- `MYSQL_OPT_LOCAL_INFILE` (argument type: optional pointer to `unsigned int`)
If no pointer is given or if pointer points to an `unsigned int` that has a non-zero value, the `LOAD LOCAL INFILE` statement is enabled.
- `MYSQL_OPT_NAMED_PIPE` (argument: not used)
Use named pipes to connect to a MySQL server on Windows, if the server allows named-pipe connections.
- `MYSQL_OPT_PROTOCOL` (argument type: `unsigned int *`)
Type of protocol to use. Should be one of the enum values of `mysql_protocol_type` defined in `mysql.h`.

- `MYSQL_OPT_READ_TIMEOUT` (argument type: `unsigned int *`)

Timeout for reads from server (works only for TCP/IP connections, and only for Windows prior to MySQL 5.1.12). You can this option so that a lost connection can be detected earlier than the TCP/IP `Close_Wait_Timeout` value of 10 minutes.

- `MYSQL_OPT_RECONNECT` (argument type: `my_bool *`)

Enable or disable automatic reconnection to the server if the connection is found to have been lost. Reconnect is off by default; this option provides a way to set reconnection behavior explicitly.

Note: `mysql_real_connect()` incorrectly reset the `MYSQL_OPT_RECONNECT` option to its default value before MySQL 5.1.6. Therefore, prior to that version, if you want reconnect to be enabled for each connection, you must call `mysql_options()` with the `MYSQL_OPT_RECONNECT` option after each call to `mysql_real_connect()`. This is not necessary as of 5.1.6: Call `mysql_options()` only before `mysql_real_connect()` as usual.

- `MYSQL_OPT_SET_CLIENT_IP` (argument type: `char *`)

For an application linked against the `libmysql` embedded server library (when `libmysql` is compiled with authentication support), this means that the user is considered to have connected from the specified IP address (specified as a string) for authentication purposes. This option is ignored for applications linked against the `libmysqlclient` client library.

- `MYSQL_OPT_SSL_VERIFY_SERVER_CERT` (argument type: `my_bool *`)

Enable or disable verification of the server's Common Name value in its certificate against the hostname used when connecting to the server. The connection is rejected if there is a mismatch. This feature can be used to prevent man-in-the-middle attacks. Verification is disabled by default. Added in MySQL 5.1.11.

- `MYSQL_OPT_USE_EMBEDDED_CONNECTION` (argument: not used)

For an application linked against the `libmysql` embedded server library, this forces the use of the embedded server for the connection. This option is ignored for applications linked against the `libmysqlclient` client library.

- `MYSQL_OPT_USE_REMOTE_CONNECTION` (argument: not used)

For an application linked against the `libmysql` embedded server library, this forces the use of a remote server for the connection. This option is ignored for applications linked against the `libmysqlclient` client library.

- `MYSQL_OPT_USE_RESULT` (argument: not used)

This option is unused.

- `MYSQL_OPT_WRITE_TIMEOUT` (argument type: `unsigned int *`)

Timeout for writes to server (works currently only on Windows on TCP/IP connections).

- `MYSQL_READ_DEFAULT_FILE` (argument type: `char *`)

Read options from the named option file instead of from `my.cnf`.

- `MYSQL_READ_DEFAULT_GROUP` (argument type: `char *`)

Read options from the named group from `my.cnf` or the file specified with `MYSQL_READ_DEFAULT_FILE`.

- `MYSQL_REPORT_DATA_TRUNCATION` (argument type: `my_bool *`)

Enable or disable reporting of data truncation errors for prepared statements via `MYSQL_BIND.error`. (Default: enabled.)

- `MYSQL_SECURE_AUTH` (argument type: `my_bool*`)

Whether to connect to a server that does not support the password hashing used in MySQL 4.1.1 and later.

- `MYSQL_SET_CHARSET_DIR` (argument type: `char*`)

The pathname to the directory that contains character set definition files.

- `MYSQL_SET_CHARSET_NAME` (argument type: `char*`)

The name of the character set to use as the default character set.

- `MYSQL_SHARED_MEMORY_BASE_NAME` (argument type: `char*`)

The name of the shared-memory object for communication to the server on Windows, if the server supports shared-memory connection. Should have the same value as the `--shared-memory-base-name` option used for the `mysqld` server you want to connect to.

Note that the `client` group is always read if you use `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP`.

The specified group in the option file may contain the following options:

Option	Description
<code>connect-timeout</code>	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.
<code>compress</code>	Use the compressed client/server protocol.
<code>database</code>	Connect to this database if no database was specified in the connect command.
<code>debug</code>	Debug options.
<code>disable-local-infile</code>	Disable use of <code>LOAD DATA LOCAL</code> .
<code>host</code>	Default hostname.
<code>init-command</code>	Statement to execute when connecting to MySQL server. Automatically re-executed if reconnection occurs.
<code>interactive-timeout</code>	Same as specifying <code>CLIENT_INTERACTIVE</code> to <code>mysql_real_connect()</code> . See Section 25.2.3.52 , “ <code>mysql_real_connect()</code> ”.
<code>local-infile[=(0 1)]</code>	If no argument or argument <code>!= 0</code> then enable use of <code>LOAD DATA LOCAL</code> .
<code>max_allowed_packet</code>	Max size of packet client can read from server.
<code>multi-results</code>	Allow multiple result sets from multiple-statement executions or stored procedures.
<code>multi-statements</code>	Allow the client to send multiple statements in a single string (separated by ‘;’).
<code>password</code>	Default password.
<code>pipe</code>	Use named pipes to connect to a MySQL server on Windows.
<code>protocol={TCP SOCKET PIPE MEMORY}</code>	The protocol to use when connecting to the server.
<code>port</code>	Default port number.
<code>return-found-rows</code>	Tell <code>mysql_info()</code> to return found rows instead of updated rows when using <code>UPDATE</code> .
<code>na shared-memory-base-name=me</code>	Shared-memory name to use to connect to server (default is “MYSQL”).
<code>socket</code>	Default socket file.
<code>user</code>	Default user.

Note that `timeout` has been replaced by `connect-timeout`, but `timeout` is still supported in MySQL 5.1 for backward compatibility.

For more information about option files, see [Section 4.3.2](#), “Using Option Files”.

Before MySQL 5.1.18, the `arg` argument was declared as `char *`.

Return Values

Zero for success. Non-zero if you specify an unknown option.

Example

```
MYSQL mysql;
```

```
mysql_init(&mysql);
mysql_options(&mysql, MYSQL_OPT_COMPRESS, 0);
mysql_options(&mysql, MYSQL_READ_DEFAULT_GROUP, "odbc");
if (!mysql_real_connect(&mysql, "host", "user", "passwd", "database", 0, NULL, 0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
        mysql_error(&mysql));
}
```

This code requests that the client use the compressed client/server protocol and read the additional options from the `odbc` section in the `my.cnf` file.

25.2.3.50. `mysql_ping()`

```
int mysql_ping(MYSQL *mysql)
```

Description

Checks whether the connection to the server is working. If the connection has gone down, an attempt to reconnect is made unless auto-reconnect is disabled.

This function can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

Return Values

Zero if the connection to the server is alive. Non-zero if an error occurred. A non-zero return does not indicate whether the MySQL server itself is down; the connection might be broken for other reasons such as network problems.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

25.2.3.51. `mysql_query()`

```
int mysql_query(MYSQL *mysql, const char *stmt_str)
```

Description

Executes the SQL statement pointed to by the null-terminated string `stmt_str`. Normally, the string must consist of a single SQL statement and you should not add a terminating semicolon (`;`) or `\g` to the statement. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See [Section 25.2.9, “C API Handling of Multiple Statement Execution”](#).

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the `'\0'` character, which `mysql_query()` interprets as the end of the statement string.)

If you want to know whether the statement should return a result set, you can use `mysql_field_count()` to check for this. See [Section 25.2.3.22, “`mysql\_field\_count\(\)`”](#).

Return Values

Zero if the statement was successful. Non-zero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC](#)

Commands were executed in an improper order.

- [CR_SERVER_GONE_ERROR](#)

The MySQL server has gone away.

- [CR_SERVER_LOST](#)

The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR](#)

An unknown error occurred.

25.2.3.52. [mysql_real_connect\(\)](#)

`MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd, const char *db, unsigned int port, const char *unix_socket, unsigned long client_flag)`

Description

[mysql_real_connect\(\)](#) attempts to establish a connection to a MySQL database engine running on [host](#).

[mysql_real_connect\(\)](#) must complete successfully before you can execute any other API functions that require a valid `MYSQL` connection handle structure.

The parameters are specified as follows:

- The first parameter should be the address of an existing `MYSQL` structure. Before calling [mysql_real_connect\(\)](#) you must call [mysql_init\(\)](#) to initialize the `MYSQL` structure. You can change a lot of connect options with the [mysql_options\(\)](#) call. See [Section 25.2.3.49, “mysql_options\(\)”](#).
- The value of [host](#) may be either a hostname or an IP address. If [host](#) is `NULL` or the string `"localhost"`, a connection to the local host is assumed: For Windows, the client connects using a shared-memory connection, if the server has shared-memory connections enabled. Otherwise, TCP/IP is used. For Unix, the client connects using a Unix socket file. For local connections, you can also influence the type of connection to use with the `MYSQL_OPT_PROTOCOL` or `MYSQL_OPT_NAMED_PIPE` options to [mysql_options\(\)](#). The type of connection must be supported by the server. For a [host](#) value of `."` on Windows, the client connects using a named pipe, if the server has named-pipe connections enabled. If named-pipe connections are not enabled, an error occurs.
- The [user](#) parameter contains the user's MySQL login ID. If [user](#) is `NULL` or the empty string `""`, the current user is assumed. Under Unix, this is the current login name. Under Windows ODBC, the current username must be specified explicitly. See the MySQL ODBC section of [Chapter 26, Connectors](#).
- The [passwd](#) parameter contains the password for [user](#). If [passwd](#) is `NULL`, only entries in the [user](#) table for the user that have a blank (empty) password field are checked for a match. This allows the database administrator to set up the MySQL privilege system in such a way that users get different privileges depending on whether they have specified a password.

Note

Do not attempt to encrypt the password before calling [mysql_real_connect\(\)](#); password encryption is handled automatically by the client API.

- The [user](#) and [passwd](#) parameters use whatever character set has been configured for the `MYSQL` object. By default, this is `latin1`, but can be changed by calling [mysql_options\(mysql, MYSQL_SET_CHARSET_NAME, "charset_name"\)](#) prior to connecting.
- [db](#) is the database name. If [db](#) is not `NULL`, the connection sets the default database to this value.

- If `port` is not 0, the value is used as the port number for the TCP/IP connection. Note that the `host` parameter determines the type of the connection.
- If `unix_socket` is not `NULL`, the string specifies the socket or named pipe that should be used. Note that the `host` parameter determines the type of the connection.
- The value of `client_flag` is usually 0, but can be set to a combination of the following flags to enable certain features:

Flag Name	Flag Description
<code>CLIENT_COMPRESS</code>	Use compression protocol.
<code>CLIENT_FOUND_ROWS</code>	Return the number of found (matched) rows, not the number of changed rows.
<code>CLIENT_IGNORE_SPACE</code>	Allow spaces after function names. Makes all functions names reserved words.
<code>CLIENT_INTERACTIVE</code>	Allow <code>interactive_timeout</code> seconds (instead of <code>wait_timeout</code> seconds) of inactivity before closing the connection. The client's session <code>wait_timeout</code> variable is set to the value of the session <code>interactive_timeout</code> variable.
<code>CLIENT_LOCAL_FILES</code>	Enable <code>LOAD DATA LOCAL</code> handling.
<code>CLIENT_MULTI_RESULTS</code>	Tell the server that the client can handle multiple result sets from multiple-statement executions or stored procedures. This is automatically set if <code>CLIENT_MULTI_STATEMENTS</code> is set. See the note following this table for more information about this flag.
<code>CLIENT_MULTI_STATEMENTS</code>	Tell the server that the client may send multiple statements in a single string (separated by <code>;</code>). If this flag is not set, multiple-statement execution is disabled. See the note following this table for more information about this flag.
<code>CLIENT_NO_SCHEMA</code>	Don't allow the <code>db_name.tbl_name.col_name</code> syntax. This is for ODBC. It causes the parser to generate an error if you use that syntax, which is useful for trapping bugs in some ODBC programs.
<code>CLIENT_ODBC</code>	Unused.
<code>CLIENT_SSL</code>	Use SSL (encrypted protocol). This option should not be set by application programs; it is set internally in the client library. Instead, use <code>mysql_ssl_set()</code> before calling <code>mysql_real_connect()</code> .

If your program uses the `CALL` SQL statement to execute stored procedures that produce result sets, you *must* set the `CLIENT_MULTI_RESULTS` flag, either explicitly, or implicitly by setting `CLIENT_MULTI_STATEMENTS` when you call `mysql_real_connect()`. This is because each such stored procedure produces multiple results: the result sets returned by statements executed within the procedure, as well as a result to indicate the call status.

If you enable `CLIENT_MULTI_STATEMENTS` or `CLIENT_MULTI_RESULTS`, you should process the result for every call to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see [Section 25.2.9, “C API Handling of Multiple Statement Execution”](#).

For some parameters, it is possible to have the value taken from an option file rather than from an explicit value in the `mysql_real_connect()` call. To do this, call `mysql_options()` with the `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP` option before calling `mysql_real_connect()`. Then, in the `mysql_real_connect()` call, specify the “no-value” value for each parameter to be read from an option file:

- For `host`, specify a value of `NULL` or the empty string (`""`).
- For `user`, specify a value of `NULL` or the empty string.
- For `passwd`, specify a value of `NULL`. (For the password, a value of the empty string in the `mysql_real_connect()` call cannot be overridden in an option file, because the empty string indicates explicitly that the MySQL account must have an empty password.)
- For `db`, specify a value of `NULL` or the empty string.
- For `port`, specify a value of 0.
- For `unix_socket`, specify a value of `NULL`.

If no value is found in an option file for a parameter, its default value is used as indicated in the descriptions given earlier in this section.

Return Values

A `MYSQL*` connection handle if the connection was successful, `NULL` if the connection was unsuccessful. For a successful connection, the return value is the same as the value of the first parameter.

Errors

- `CR_CONN_HOST_ERROR`

Failed to connect to the MySQL server.

- `CR_CONNECTION_ERROR`

Failed to connect to the local MySQL server.

- `CR_IPSOCK_ERROR`

Failed to create an IP socket.

- `CR_OUT_OF_MEMORY`

Out of memory.

- `CR_SOCKET_CREATE_ERROR`

Failed to create a Unix socket.

- `CR_UNKNOWN_HOST`

Failed to find the IP address for the hostname.

- `CR_VERSION_ERROR`

A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version. This can happen if you use a very old client library to connect to a new server that wasn't started with the `--old-protocol` option.

- `CR_NAMEDPIPEOPEN_ERROR`

Failed to create a named pipe on Windows.

- `CR_NAMEDPIPEWAIT_ERROR`

Failed to wait for a named pipe on Windows.

- `CR_NAMEDPIPESETSTATE_ERROR`

Failed to get a pipe handler on Windows.

- `CR_SERVER_LOST`

If `connect_timeout > 0` and it took longer than `connect_timeout` seconds to connect to the server or if the server died while executing the `init-command`.

Example

```
MYSQL mysql;
mysql_init(&mysql);
mysql_options(&mysql, MYSQL_READ_DEFAULT_GROUP, "your_prog_name");
if (!mysql_real_connect(&mysql, "host", "user", "passwd", "database", 0, NULL, 0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
            mysql_error(&mysql));
}
```


By using `mysql_options()` the MySQL library reads the `[client]` and `[your_prog_name]` sections in the `my.cnf` file which ensures that your program works, even if someone has set up MySQL in some non-standard way.

Note that upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API older than 5.0.3, or `0` in newer versions. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. You can use the `MYSQL_OPT_RECONNECT` option to `mysql_options()` to control reconnection behavior.

25.2.3.53. `mysql_real_escape_string()`

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char *from, unsigned long length)
```

Note that `mysql` must be a valid, open connection. This is needed because the escaping depends on the character set in use by the server.

Description

This function is used to create a legal SQL string that you can use in an SQL statement. See [Section 8.1.1, “Strings”](#).

The string in `from` is encoded to an escaped SQL string, taking into account the current character set of the connection. The result is placed in `to` and a terminating null byte is appended. Characters encoded are `NUL` (ASCII 0), `'\n'`, `'\r'`, `'\'`, `'\"'`, and Control-Z (see [Section 8.1, “Literal Values”](#)). (Strictly speaking, MySQL requires only that backslash and the quote character used to quote the string in the query be escaped. This function quotes the other characters to make them easier to read in log files.)

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. (In the worst case, each character may need to be encoded as using two bytes, and you need room for the terminating null byte.) When `mysql_real_escape_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

If you need to change the character set of the connection, you should use the `mysql_set_character_set()` function rather than executing a `SET NAMES` (or `SET CHARACTER SET`) statement. `mysql_set_character_set()` works like `SET NAMES` but also affects the character set used by `mysql_real_escape_string()`, which `SET NAMES` does not.

Example

```
char query[1000], *end;
end = strmov(query, "INSERT INTO test_table values(");
*end++ = '\n';
end += mysql_real_escape_string(&mysql, end, "What's this", 11);
*end++ = '\n';
*end++ = '\n';
*end++ = '\n';
end += mysql_real_escape_string(&mysql, end, "binary data: \0\r\n", 16);
*end++ = '\n';
*end++ = ')';

if (mysql_real_query(&mysql, query, (unsigned int) (end - query)))
{
    fprintf(stderr, "Failed to insert row, Error: %s\n",
        mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

25.2.3.54. `mysql_real_query()`

```
int mysql_real_query(MYSQL *mysql, const char *stmt_str, unsigned long length)
```

Description

Executes the SQL statement pointed to by `stmt_str`, which should be a string `length` bytes long. Normally, the string must consist of a single SQL statement and you should not add a terminating semicolon (`;`) or `\g` to the statement. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See [Section 25.2.9, “C API Handling of Multiple Statement Execution”](#).

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the `'\0'` character, which `mysql_query()` interprets as the end of the statement string.) In addition, `mysql_real_query()` is faster than `mysql_query()` because it does not call `strlen()` on the statement string.

If you want to know whether the statement should return a result set, you can use `mysql_field_count()` to check for this. See [Section 25.2.3.22, “`mysql\_field\_count\(\)`”](#).

Return Values

Zero if the statement was successful. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

25.2.3.55. `mysql_refresh()`

```
int mysql_refresh(MYSQL *mysql, unsigned int options)
```

Description

This function flushes tables or caches, or resets replication server information. The connected user must have the `RELOAD` privilege.

The `options` argument is a bit mask composed from any combination of the following values. Multiple values can be OR'ed together to perform multiple operations with a single call.

- `REFRESH_GRANT`
Refresh the grant tables, like `FLUSH PRIVILEGES`.
- `REFRESH_LOG`
Flush the logs, like `FLUSH LOGS`.
- `REFRESH_TABLES`
Flush the table cache, like `FLUSH TABLES`.
- `REFRESH_HOSTS`
Flush the host cache, like `FLUSH HOSTS`.

- [REFRESH_STATUS](#)
Reset status variables, like [FLUSH STATUS](#).
- [REFRESH_THREADS](#)
Flush the thread cache.
- [REFRESH_SLAVE](#)
On a slave replication server, reset the master server information and restart the slave, like [RESET SLAVE](#).
- [REFRESH_MASTER](#)
On a master replication server, remove the binary log files listed in the binary log index and truncate the index file, like [RESET MASTER](#).

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR](#)
The MySQL server has gone away.
- [CR_SERVER_LOST](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR](#)
An unknown error occurred.

25.2.3.56. [mysql_reload\(\)](#)

```
int mysql_reload(MYSQL *mysql)
```

Description

Asks the MySQL server to reload the grant tables. The connected user must have the [RELOAD](#) privilege.

This function is deprecated. It is preferable to use [mysql_query\(\)](#) to issue an SQL [FLUSH PRIVILEGES](#) statement instead.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC](#)
Commands were executed in an improper order.
- [CR_SERVER_GONE_ERROR](#)
The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

25.2.3.57. `mysql_rollback()`

```
my_bool mysql_rollback(MYSQL *mysql)
```

Description

Rolls back the current transaction.

The action of this function is subject to the value of the `completion_type` system variable. In particular, if the value of `completion_type` is 2, the server performs a release after terminating a transaction and closes the client connection. The client program should call `mysql_close()` to close the connection from the client side.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

25.2.3.58. `mysql_row_seek()`

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a query result set. The `offset` value is a row offset that should be a value returned from `mysql_row_tell()` or from `mysql_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_row_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

Errors

None.

25.2.3.59. `mysql_row_tell()`

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

Description

Returns the current position of the row cursor for the last `mysql_fetch_row()`. This value can be used as an argument to `mysql_row_seek()`.

You should use `mysql_row_tell()` only after `mysql_store_result()`, not after `mysql_use_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

25.2.3.60. `mysql_select_db()`

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

Description

Causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

`mysql_select_db()` fails unless the connected user can be authenticated as having permission to use the database.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

25.2.3.61. `mysql_set_character_set()`

```
int mysql_set_character_set(MYSQL *mysql, char *csname)
```

Description

This function is used to set the default character set for the current connection. The string `csname` specifies a valid character set name. The connection collation becomes the default collation of the character set. This function works like the `SET NAMES` statement, but also sets the value of `mysql->charset`, and thus affects the character set used by `mysql_real_escape_string()`

Return Values

Zero for success. Non-zero if an error occurred.

Example

```
MYSQL mysql;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql, "host", "user", "passwd", "database", 0, NULL, 0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
            mysql_error(&mysql));
}

if (!mysql_set_character_set(&mysql, "utf8"))
{
    printf("New client character set: %s\n",
            mysql_character_set_name(&mysql));
}
```

25.2.3.62. `mysql_set_local_infile_default()`

```
void
mysql_set_local_infile_default(MYSQL *mysql);
```

Description

Sets the **LOAD LOCAL DATA INFILE** handler callback functions to the defaults used internally by the C client library. The library calls this function automatically if `mysql_set_local_infile_handler()` has not been called or does not supply valid functions for each of its callbacks.

The `mysql_set_local_infile_default()` function was added in MySQL 4.1.2.

Return Values

None.

Errors

None.

25.2.3.63. `mysql_set_local_infile_handler()`

```
void
mysql_set_local_infile_handler(MYSQL *mysql,
    int (*local_infile_init)(void **, const char *, void *),
    int (*local_infile_read)(void *, char *, unsigned int),
    void (*local_infile_end)(void *),
    int (*local_infile_error)(void *, char*, unsigned int),
    void *userdata);
```

Description

This function installs callbacks to be used during the execution of **LOAD DATA LOCAL INFILE** statements. It enables application programs to exert control over local (client-side) data file reading. The arguments are the connection handler, a set of pointers to callback functions, and a pointer to a data area that the callbacks can use to share information.

To use `mysql_set_local_infile_handler()`, you must write the following callback functions:

```
int
local_infile_init(void **ptr, const char *filename, void *userdata);
```

The initialization function. This is called once to do any setup necessary, open the data file, allocate data structures, and so forth. The first `void**` argument is a pointer to a pointer. You can set the pointer (that is, `*ptr`) to a value that will be passed to each of the other callbacks (as a `void*`). The callbacks can use this pointed-to value to maintain state information. The `userdata` argument is the same value that is passed to `mysql_set_local_infile_handler()`.

The initialization function should return zero for success, non-zero for an error.

```
int
local_infile_read(void *ptr, char *buf, unsigned int buf_len);
```

The data-reading function. This is called repeatedly to read the data file. `buf` points to the buffer where the read data should be stored, and `buf_len` is the maximum number of bytes that the callback can read and store in the buffer. (It can read fewer bytes, but should not read more.)

The return value is the number of bytes read, or zero when no more data could be read (this indicates EOF). Return a value less than zero if an error occurs.

```
void
local_infile_end(void *ptr)
```

The termination function. This is called once after `local_infile_read()` has returned zero (EOF) or an error. This function should deallocate any memory allocated by `local_infile_init()` and perform any other cleanup necessary. It is invoked even if the initialization function returns an error.

```
int
local_infile_error(void *ptr,
    char *error_msg,
```

```
unsigned int error_msg_len);
```

The error-handling function. This is called to get a textual error message to return to the user in case any of your other functions returns an error. `error_msg` points to the buffer into which the message should be written, and `error_msg_len` is the length of the buffer. The message should be written as a null-terminated string, so the message can be at most `error_msg_len-1` bytes long.

The return value is the error number.

Typically, the other callbacks store the error message in the data structure pointed to by `ptr`, so that `local_infile_error()` can copy the message from there into `error_msg`.

After calling `mysql_set_local_infile_handler()` in your C code and passing pointers to your callback functions, you can then issue a `LOAD DATA LOCAL INFILE` statement (for example, by using `mysql_query()`). The client library automatically invokes your callbacks. The filename specified in `LOAD DATA LOCAL INFILE` will be passed as the second parameter to the `local_infile_init()` callback.

The `mysql_set_local_infile_handler()` function was added in MySQL 4.1.2.

Return Values

None.

Errors

None.

25.2.3.64. `mysql_set_server_option()`

```
int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)
```

Description

Enables or disables an option for the connection. `option` can have one of the following values:

MYSQL_OPTION_MULTI_STATEMENTS_ON	Enable multiple-statement support.
MYSQL_OPTION_MULTI_STATEMENTS_OFF	Disable multiple-statement support.

If you enable multiple-statement support, you should retrieve results from calls to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see [Section 25.2.9, “C API Handling of Multiple Statement Execution”](#).

Enabling multiple-statement support with `MYSQL_OPTION_MULTI_STATEMENTS_ON` does not have quite the same effect as enabling it by passing the `CLIENT_MULTI_STATEMENTS` flag to `mysql_real_connect()`: `CLIENT_MULTI_STATEMENTS` also enables `CLIENT_MULTI_RESULTS`. If you are using the `CALL SQL` statement in your programs, multiple-result support must be enabled; this means that `MYSQL_OPTION_MULTI_STATEMENTS_ON` by itself is insufficient to allow the use of `CALL`.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `ER_UNKNOWN_COM_ERROR`

The server didn't support `mysql_set_server_option()` (which is the case that the server is older than 4.1.1) or the server didn't support the option one tried to set.

25.2.3.65. `mysql_shutdown()`

```
int mysql_shutdown(MYSQL *mysql, enum mysql_enum_shutdown_level shutdown_level)
```

Description

Asks the database server to shut down. The connected user must have `SHUTDOWN` privileges. MySQL 5.1 servers support only one type of shutdown; `shutdown_level` must be equal to `SHUTDOWN_DEFAULT`. Additional shutdown levels are planned to make it possible to choose the desired level. Dynamically linked executables which have been compiled with older versions of the `libmysql-client` headers and call `mysql_shutdown()` need to be used with the old `libmysqlclient` dynamic library.

The shutdown process is described in [Section 5.2.7, “The Shutdown Process”](#).

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

25.2.3.66. `mysql_sqlstate()`

```
const char *mysql_sqlstate(MYSQL *mysql)
```

Description

Returns a null-terminated string containing the SQLSTATE error code for the most recently executed SQL statement. The error code consists of five characters. `'00000'` means “no error”. The values are specified by ANSI SQL and ODBC. For a list of possible values, see [Appendix B, Errors, Error Codes, and Common Problems](#).

SQLSTATE values returned by `mysql_sqlstate()` differ from MySQL-specific error numbers returned by `mysql_errno()`. For example, the `mysql` client program displays errors using the following format, where `1146` is the `mysql_errno()` value and `'42S02'` is the corresponding `mysql_sqlstate()` value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Not all MySQL error numbers are mapped to SQLSTATE error codes. The value `'HY000'` (general error) is used for unmapped error numbers.

If you call `mysql_sqlstate()` after `mysql_real_connect()` fails, `mysql_sqlstate()` might not return a useful value. For example, this happens if a host is blocked by the server and the connection is closed without any SQLSTATE value being sent to the client.

Return Values

A null-terminated character string containing the SQLSTATE error code.

See Also

See [Section 25.2.3.14](#), “`mysql_errno()`”, [Section 25.2.3.15](#), “`mysql_error()`”, and [Section 25.2.7.26](#), “`mysql_stmt_sqlstate()`”.

25.2.3.67. `mysql_ssl_set()`

```
int mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert, const char *ca, const char *capath, const char *cipher)
```

Description

`mysql_ssl_set()` is used for establishing secure connections using SSL. It must be called before `mysql_real_connect()`.

`mysql_ssl_set()` does nothing unless OpenSSL support is enabled in the client library.

`mysql` is the connection handler returned from `mysql_init()`. The other parameters are specified as follows:

- `key` is the pathname to the key file.
- `cert` is the pathname to the certificate file.
- `ca` is the pathname to the certificate authority file.
- `capath` is the pathname to a directory that contains trusted SSL CA certificates in pem format.
- `cipher` is a list of allowable ciphers to use for SSL encryption.

Any unused SSL parameters may be given as `NULL`.

Return Values

This function always returns 0. If SSL setup is incorrect, `mysql_real_connect()` returns an error when you attempt to connect.

25.2.3.68. `mysql_stat()`

```
const char *mysql_stat(MYSQL *mysql)
```

Description

Returns a character string containing information similar to that provided by the `mysqladmin status` command. This includes up-time in seconds and the number of running threads, questions, reloads, and open tables.

Return Values

A character string describing the server status. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

25.2.3.69. `mysql_store_result()`

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Description

After invoking `mysql_query()` or `mysql_real_query()`, you must call `mysql_store_result()` or `mysql_use_result()` for every statement that successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth). You must also call `mysql_free_result()` after you are done with the result set.

You don't have to call `mysql_store_result()` or `mysql_use_result()` for other statements, but it does not do any harm or cause any notable performance degradation if you call `mysql_store_result()` in all cases. You can detect whether the statement has a result set by checking whether `mysql_store_result()` returns a non-zero value (more about this later on).

If you enable multiple-statement support, you should retrieve results from calls to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see [Section 25.2.9, “C API Handling of Multiple Statement Execution”](#).

If you want to know whether a statement should return a result set, you can use `mysql_field_count()` to check for this. See [Section 25.2.3.22, “mysql_field_count\(\)”](#).

`mysql_store_result()` reads the entire result of a query to the client, allocates a `MYSQL_RES` structure, and places the result into this structure.

`mysql_store_result()` returns a null pointer if the statement didn't return a result set (for example, if it was an `INSERT` statement).

`mysql_store_result()` also returns a null pointer if reading of the result set failed. You can check whether an error occurred by checking whether `mysql_error()` returns a non-empty string, `mysql_errno()` returns non-zero, or `mysql_field_count()` returns zero.

An empty result set is returned if there are no rows returned. (An empty result set differs from a null pointer as a return value.)

After you have called `mysql_store_result()` and gotten back a result that isn't a null pointer, you can call `mysql_num_rows()` to find out how many rows are in the result set.

You can call `mysql_fetch_row()` to fetch rows from the result set, or `mysql_row_seek()` and `mysql_row_tell()` to obtain or set the current row position within the result set.

See [Section 25.2.14.1, “Why mysql_store_result\(\) Sometimes Returns NULL After mysql_query\(\) Returns Success”](#).

Return Values

A `MYSQL_RES` result structure with the results. `NULL` (0) if an error occurred.

Errors

`mysql_store_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

Out of memory.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

25.2.3.70. `mysql_thread_id()`

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

Description

Returns the thread ID of the current connection. This value can be used as an argument to `mysql_kill()` to kill the thread.

If the connection is lost and you reconnect with `mysql_ping()`, the thread ID changes. This means you should not get the thread ID and store it for later. You should get it when you need it.

Return Values

The thread ID of the current connection.

Errors

None.

25.2.3.71. `mysql_use_result()`

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query that successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`).

`mysql_use_result()` initiates a result set retrieval but does not actually read the result set into the client like `mysql_store_result()` does. Instead, each row must be retrieved individually by making calls to `mysql_fetch_row()`. This reads the result of a query directly from the server without storing it in a temporary table or local buffer, which is somewhat faster and uses much less memory than `mysql_store_result()`. The client allocates memory only for the current row and a communication buffer that may grow up to `max_allowed_packet` bytes.

On the other hand, you shouldn't use `mysql_use_result()` if you are doing a lot of processing for each row on the client side, or if the output is sent to a screen on which the user may type a `^S` (stop scroll). This ties up the server and prevent other threads from updating any tables from which the data is being fetched.

When using `mysql_use_result()`, you must execute `mysql_fetch_row()` until a `NULL` value is returned, otherwise, the un-fetched rows are returned as part of the result set for your next query. The C API gives the error `Commands out of sync; you can't run this command now` if you forget to do this!

You may not use `mysql_data_seek()`, `mysql_row_seek()`, `mysql_row_tell()`, `mysql_num_rows()`, or `mysql_affected_rows()` with a result returned from `mysql_use_result()`, nor may you issue other queries until `mysql_use_result()` has finished. (However, after you have fetched all the rows, `mysql_num_rows()` accurately returns the number of rows fetched.)

You must call `mysql_free_result()` once you are done with the result set.

When using the `libmysql` embedded server, the memory benefits are essentially lost because memory usage incrementally increases with each row retrieved until `mysql_free_result()` is called.

Return Values

A `MYSQL_RES` result structure. `NULL` if an error occurred.

Errors

`mysql_use_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_OUT_OF_MEMORY`
Out of memory.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

25.2.3.72. `mysql_warning_count()`

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

Description

Returns the number of warnings generated during execution of the previous SQL statement.

Return Values

The warning count.

Errors

None.

25.2.4. C API Prepared Statements

The MySQL client/server protocol provides for the use of prepared statements. This capability uses the `MYSQL_STMT` statement handler data structure returned by the `mysql_stmt_init()` initialization function. Prepared execution is an efficient way to execute a statement more than once. The statement is first parsed to prepare it for execution. Then it is executed one or more times at a later time, using the statement handle returned by the initialization function.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Prepared statements might not provide a performance increase in some situations. For best results, test your application both with prepared and non-prepared statements and choose whichever yields best performance.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

The following statements can be used as prepared statements: `CREATE`, `TABLE`, `DELETE`, `DO`, `INSERT`, `REPLACE`, `SELECT`, `SET`, `UPDATE`, and most `SHOW` statements.

As of MySQL 5.1.10, the following additional statements are supported:

```
ANALYZE TABLE
OPTIMIZE TABLE
REPAIR TABLE
```

As of MySQL 5.1.12, the following additional statements are supported:

```
CACHE INDEX
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
{CREATE | RENAME | DROP} DATABASE
{CREATE | RENAME | DROP} USER
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
      | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES}
GRANT
REVOKE
KILL
LOAD INDEX INTO CACHE
RESET {MASTER | SLAVE | QUERY CACHE}
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {AUTHORS | CONTRIBUTORS | WARNINGS | ERRORS}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
INSTALL PLUGIN
UNINSTALL PLUGIN
```

Other statements are not yet supported in MySQL 5.1.

MySQL Enterprise

MySQL Enterprise subscribers will find more information about using prepared statements in the Knowledge Base article, [How can I create server-side prepared statements?](https://www.mysql.com/knowledge-base/how-can-i-create-server-side-prepared-statements/). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

25.2.5. C API Prepared Statement Data types

Prepared statements use several data structures:

- To prepare a statement, pass the statement string to `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT` data structure.
- To provide input parameters for a prepared statement, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_param()`. To receive output column values, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_result()`.
- The `MYSQL_TIME` structure is used to transfer temporal data in both directions.

The following discussion describes the prepared statement data types in detail.

`MYSQL_STMT`

This structure represents a prepared statement. A statement is created by calling `mysql_stmt_init()`, which returns a statement handle (that is, a pointer to a `MYSQL_STMT`). The handle is used for all subsequent operations with the statement until you close it with `mysql_stmt_close()`, at which point the handle becomes invalid.

The `MYSQL_STMT` structure has no members that are intended for application use. Also, you should not try to make a copy of a `MYSQL_STMT` structure. There is no guarantee that such a copy will be usable.

Multiple statement handles can be associated with a single connection. The limit on the number of handles depends on the available system resources.

- **MYSQL_BIND**

This structure is used both for statement input (data values sent to the server) and output (result values returned from the server):

- For input, **MYSQL_BIND** is used with `mysql_stmt_bind_param()` to bind parameter data values to buffers for use by `mysql_stmt_execute()`.
- For output, **MYSQL_BIND** is used with `mysql_stmt_bind_result()` to bind result set buffers for use in fetching rows with `mysql_stmt_fetch()`.

To use a **MYSQL_BIND** structure, you should zero its contents to initialize it, and then set its members appropriately. For example, to declare and initialize an array of three **MYSQL_BIND** structures, use this code:

```
MYSQL_BIND bind[3];
memset(bind, 0, sizeof(bind));
```

The **MYSQL_BIND** structure contains the following members for use by application programs. For several of the members, the manner of use depends on whether the structure is used for input or output.

- **enum enum_field_types buffer_type**

The type of the buffer. This member indicates the data type of the C language variable that you are binding to the statement parameter. The allowable **buffer_type** values are listed later in this section. For input, **buffer_type** indicates the type of the variable containing the value that you will send to the server. For output, it indicates the type of the variable into which you want a value received from the server to be stored.

- **void *buffer**

A pointer to the buffer to be used for data transfer. This is the address of a variable.

For input, **buffer** is a pointer to the variable in which a statement parameter's data value is stored. When you call `mysql_stmt_execute()`, MySQL takes the value that you have stored in the variable and uses it in place of the corresponding parameter marker in the statement.

For output, **buffer** is a pointer to the variable in which to return a result set column value. When you call `mysql_stmt_fetch()`, MySQL returns a column value and stores it in this variable. You can access the value when the call returns.

To minimize the need for MySQL to perform type conversions between C language values on the client side and SQL values on the server side, use variables that have types similar to those of the corresponding SQL values. For numeric data types, **buffer** should point to a variable of the proper numeric C type. (For **char** or integer variables, you should also indicate whether the variable has the **unsigned** attribute by setting the **is_unsigned** member, described later in this list.) For character (non-binary) and binary string data types, **buffer** should point to a character buffer. For date and time data types, **buffer** should point to a **MYSQL_TIME** structure.

See the notes about type conversions later in the section.

- **unsigned long buffer_length**

The actual size of ***buffer** in bytes. This indicates the maximum amount of data that can be stored in the buffer. For character and binary C data, the **buffer_length** value specifies the length of ***buffer** when used with `mysql_stmt_bind_param()` to specify input values, or the maximum number of output data bytes that can be fetched into the buffer when used with `mysql_stmt_bind_result()`.

- **unsigned long *length**

A pointer to an **unsigned long** variable that indicates the actual number of bytes of data stored in ***buffer**. **length** is used for character or binary C data.

For input parameter data binding, **length** points to an **unsigned long** variable that indicates the actual length of the parameter value stored in ***buffer**; this is used by `mysql_stmt_execute()`.

For output value binding, the return value of `mysql_stmt_fetch()` determines the interpretation of the length:

- If `mysql_stmt_fetch()` returns 0, ***length** indicates the actual length of the parameter value.

- If `mysql_stmt_fetch()` returns `MYSQL_DATA_TRUNCATED`, `*length` indicates the non-truncated length of the parameter value. In this case, the minimum of `*length` and `buffer_length` indicates the actual length of the value.

`length` is ignored for numeric and temporal data types because the length of the data value is determined by the `buffer_type` value.

If you need to be able to determine the length of a returned value before fetching it with `mysql_stmt_fetch()`, see [Section 25.2.7.11, “mysql_stmt_fetch\(\)”](#), for some strategies.

- `my_bool *is_null`

This member points to a `my_bool` variable that is true if a value is `NULL`, false if it is not `NULL`. For input, set `*is_null` to true to indicate that you are passing a `NULL` value as a statement parameter.

The reason that `is_null` is not a boolean scalar but is instead a *pointer* to a boolean scalar is to provide flexibility in how you specify `NULL` values:

- If your data values are always `NULL`, use `MYSQL_TYPE_NULL` as the `buffer_type` value when you bind the column. The other members do not matter.
- If your data values are always `NOT NULL`, set the other members appropriately for the variable you are binding, and set `is_null = (my_bool*) 0`.
- In all other cases, set the other members appropriately, and set `is_null` to the address of a `my_bool` variable. Set that variable's value to true or false appropriately between executions to indicate whether data values are `NULL` or `NOT NULL`, respectively.

For output, the value pointed to by `is_null` is set to true after you fetch a row if the result set column value returned from the statement is `NULL`.

- `my_bool is_unsigned`

This member is used for C variables with data types that can be `unsigned` (`char`, `short int`, `int`, `long long int`). Set `is_unsigned` to true if the variable pointed to by `buffer` is `unsigned` and false otherwise. For example, if you bind a `signed char` variable to `buffer`, specify a type code of `MYSQL_TYPE_TINY` and set `is_unsigned` to false. If you bind an `unsigned char` instead, the type code is the same but `is_unsigned` should be true. (For `char`, it is not defined whether it is signed or unsigned, so it is best to be explicit about signedness by using `signed char` or `unsigned char`.)

`is_unsigned` applies only to the C language variable on the client side. It indicates nothing about the signedness of the corresponding SQL value on the server side. For example, if you use an `int` variable to supply a value for a `BIGINT UNSIGNED` column, `is_unsigned` should be false because `int` is a signed type. If you use an `unsigned int` variable to supply a value for a `BIGINT` column, `is_unsigned` should be true because `unsigned int` is an unsigned type. MySQL performs the proper conversion between signed and unsigned values in both directions, although a warning occurs if truncation results.

- `my_bool *error`

For output, set this member to point to a `my_bool` variable to have truncation information for the parameter stored there after a row fetching operation. (Truncation reporting is enabled by default, but can be controlled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option.) When truncation reporting is enabled, `mysql_stmt_fetch()` returns `MYSQL_DATA_TRUNCATED` and `*error` is true in the `MYSQL_BIND` structures for parameters in which truncation occurred. Truncation indicates loss of sign or significant digits, or that a string was too long to fit in a column.

- `MYSQL_TIME`

This structure is used to send and receive `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` data directly to and from the server. Set the `buffer_type` member of a `MYSQL_BIND` structure to one of the temporal types (`MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATE`, `MYSQL_TYPE_DATETIME`, `MYSQL_TYPE_TIMESTAMP`), and set the `buffer` member to point to a `MYSQL_TIME` structure.

The `MYSQL_TIME` structure contains the members listed in the following table.

Member	Description
<code>unsigned int year</code>	The year
<code>unsigned int month</code>	The month of the year

<code>unsigned int day</code>	The day of the month
<code>unsigned int hour</code>	The hour of the day
<code>unsigned int minute</code>	The minute of the hour
<code>unsigned int second</code>	The second of the minute
<code>my_bool neg</code>	A boolean flag to indicate whether the time is negative
<code>unsigned long second_part</code>	The fractional part of the second in microseconds; currently unused

Only those parts of a `MYSQL_TIME` structure that apply to a given type of temporal value are used. The `year`, `month`, and `day` elements are used for `DATE`, `DATETIME`, and `TIMESTAMP` values. The `hour`, `minute`, and `second` elements are used for `TIME`, `DATETIME`, and `TIMESTAMP` values. See [Section 25.2.10, “C API Handling of Date and Time Values”](#).

The following table shows the allowable values that may be specified in the `buffer_type` member of `MYSQL_BIND` structures for input values. The value should be chosen according to the data type of the C language variable that you are binding. If the variable is `unsigned`, you should also set the `is_unsigned` member to true. The table shows the C variable types that you can use, the corresponding type codes, and the SQL data types for which the supplied value can be used without conversion.

Input Variable C Type	<code>buffer_type</code> Value	SQL Type of Destination Value
<code>signed char</code>	<code>MYSQL_TYPE_TINY</code>	<code>TINYINT</code>
<code>short int</code>	<code>MYSQL_TYPE_SHORT</code>	<code>SMALLINT</code>
<code>int</code>	<code>MYSQL_TYPE_LONG</code>	<code>INT</code>
<code>long long int</code>	<code>MYSQL_TYPE_LONGLONG</code>	<code>BIGINT</code>
<code>float</code>	<code>MYSQL_TYPE_FLOAT</code>	<code>FLOAT</code>
<code>double</code>	<code>MYSQL_TYPE_DOUBLE</code>	<code>DOUBLE</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_TIME</code>	<code>TIME</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_DATE</code>	<code>DATE</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_DATETIME</code>	<code>DATETIME</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_TIMESTAMP</code>	<code>TIMESTAMP</code>
<code>char[]</code>	<code>MYSQL_TYPE_STRING</code> (for non-binary data)	<code>TEXT</code> , <code>CHAR</code> , <code>VARCHAR</code>
<code>char[]</code>	<code>MYSQL_TYPE_BLOB</code> (for binary data)	<code>BLOB</code> , <code>BINARY</code> , <code>VARBINARY</code>
	<code>MYSQL_TYPE_NULL</code>	<code>NULL</code>

The use of `MYSQL_TYPE_NULL` is described earlier in connection with the `is_null` member.

The following table shows the allowable values that may be specified in the `buffer_type` member of `MYSQL_BIND` structures for output values. The value should be chosen according to the data type of the C language variable that you are binding. If the variable is `unsigned`, you should also set the `is_unsigned` member to true. The table shows the SQL types of received values, the corresponding type code that such values have in result set metadata, and the recommended C language data types to bind to the `MYSQL_BIND` structure to receive the SQL values without conversion.

SQL Type of Received Value	<code>buffer_type</code> Value	Output Variable C Type
<code>TINYINT</code>	<code>MYSQL_TYPE_TINY</code>	<code>signed char</code>
<code>SMALLINT</code>	<code>MYSQL_TYPE_SHORT</code>	<code>short int</code>
<code>MEDIUMINT</code>	<code>MYSQL_TYPE_INT24</code>	<code>int</code>
<code>INT</code>	<code>MYSQL_TYPE_LONG</code>	<code>int</code>
<code>BIGINT</code>	<code>MYSQL_TYPE_LONGLONG</code>	<code>long long int</code>
<code>FLOAT</code>	<code>MYSQL_TYPE_FLOAT</code>	<code>float</code>
<code>DOUBLE</code>	<code>MYSQL_TYPE_DOUBLE</code>	<code>double</code>
<code>DECIMAL</code>	<code>MYSQL_TYPE_NEWDECIMAL</code>	<code>char[]</code>

YEAR	MYSQL_TYPE_SHORT	short int
TIME	MYSQL_TYPE_TIME	MYSQL_TIME
DATE	MYSQL_TYPE_DATE	MYSQL_TIME
DATETIME	MYSQL_TYPE_DATETIME	MYSQL_TIME
TIMESTAMP	MYSQL_TYPE_TIMESTAMP	MYSQL_TIME
CHAR, BINARY	MYSQL_TYPE_STRING	char []
VARCHAR, VARBINARY	MYSQL_TYPE_VAR_STRING	char []
TINYBLOB, TINYTEXT	MYSQL_TYPE_TINY_BLOB	char []
BLOB, TEXT	MYSQL_TYPE_BLOB	char []
MEDIUMBLOB, MEDIUMTEXT	MYSQL_TYPE_MEDIUM_BLOB	char []
LONGBLOB, LONGTEXT	MYSQL_TYPE_LONG_BLOB	char []
BIT	MYSQL_TYPE_BIT	char []

The C language variable types are those recommended if you want to avoid type conversions. If there is a mismatch between the C variable type on the client side and the corresponding SQL value on the server side, MySQL performs implicit type conversions in both directions.

MySQL knows the type code for the SQL value on the server side. The `buffer_type` value indicates the MySQL the type code of the C variable that holds the value on the client side. The two codes together tell MySQL what conversion must be performed, if any. Here are some examples:

- If you use `MYSQL_TYPE_LONG` with an `int` variable to pass an integer value to the server that is to be stored into a `FLOAT` column, MySQL converts the value to floating-point format before storing it.
- If you fetch a SQL `MEDIUMINT` column value, but specify a `buffer_type` value of `MYSQL_TYPE_LONGLONG` and use a C variable of type `long long int` as the destination buffer, MySQL will convert the `MEDIUMINT` value (which requires less than 8 bytes) for storage into the `long long int` (an 8-byte variable).
- If you fetch a numeric column with a value of 255 into a `char[4]` character array and specify a `buffer_type` value of `MYSQL_TYPE_STRING`, the resulting value in the array will be a 4-byte string containing `'255\0'`.
- `DECIMAL` values are returned as strings, which is why the corresponding C type is `char[]`. `DECIMAL` values returned by the server correspond to the string representation of the original server-side value. For example, `12.345` is returned to the client as `'12.345'`. If you specify `MYSQL_TYPE_NEWDECIMAL` and bind a string buffer to the `MYSQL_BIND` structure, `mysql_stmt_fetch()` stores the value in the buffer without conversion. If instead you specify a numeric variable and type code, `mysql_stmt_fetch()` converts the string-format `DECIMAL` value to numeric form.
- For the `MYSQL_TYPE_BIT` type code, `BIT` values are returned into a string buffer (thus, the corresponding C type is `char[]` here, too). The value represents a bit string that requires interpretation on the client side. To return the value as a type that is easier to deal with, you can use a query of the following form that uses `+ 0` to cause the value to be cast to integer:

```
SELECT bit_col + 0 FROM t
```

To retrieve the value, bind an integer variable large enough to hold the value and specify the appropriate corresponding integer type code.

Before binding variables to the `MYSQL_BIND` structures that are to be used for fetching column values, you can check the type codes for each column of the result set. This might be desirable if you want to determine which variable types would be best to use to avoid type conversions. To get the type codes, call `mysql_stmt_result_metadata()` after executing the prepared statement with `mysql_stmt_execute()`. The metadata provides access to the type codes for the result set as described in [Section 25.2.7.22](#), “`mysql_stmt_result_metadata()`”, and [Section 25.2.1](#), “C API Data types”.

If you cause the `max_length` member of the `MYSQL_FIELD` column metadata structures to be set (by calling `mysql_stmt_attr_set()`), be aware that the `max_length` values for the result set indicate the lengths of the longest string representation of the result values, not the lengths of the binary representation. That is, `max_length` does not necessarily correspond to the size of the buffers needed to fetch the values with the binary protocol used for prepared statements. The size of the buffers should be chosen according to the types of the variables into which you fetch the values.

For input character (non-binary) string data (indicated by `MYSQL_TYPE_STRING`), the value is assumed to be in the character set indicated by the `character_set_client` system variable. If the value is stored into a column with a different character set, the appropriate conversion to that character set occurs. For input binary string data (indicated by `MYSQL_TYPE_BLOB`), the value is treated as having the `binary` character set; that is, it is treated as a byte string and no conversion occurs.

To determine whether output string values in a result set returned from the server contain binary or non-binary data, check whether the `charsetnr` value of the result set metadata is 63 (see [Section 25.2.1, “C API Data types”](#)). If so, the character set is `binary`, which indicates binary rather than non-binary data. This enables you to distinguish between `BINARY` and `CHAR`, `VARBINARY` and `VARCHAR`, and the `BLOB` and `TEXT` types.

25.2.6. C API Prepared Statement Function Overview

The functions available for prepared statement processing are summarized here and described in greater detail in a later section. See [Section 25.2.7, “C API Prepared Statement Function Descriptions”](#).

Function	Description
<code>mysql_stmt_affected_rows()</code>	Returns the number of rows changes, deleted, or inserted by prepared <code>UPDATE</code> , <code>DELETE</code> , or <code>INSERT</code> statement.
<code>mysql_stmt_attr_get()</code>	Get value of an attribute for a prepared statement.
<code>mysql_stmt_attr_set()</code>	Sets an attribute for a prepared statement.
<code>mysql_stmt_bind_param()</code>	Associates application data buffers with the parameter markers in a prepared SQL statement.
<code>mysql_stmt_bind_result()</code>	Associates application data buffers with columns in the result set.
<code>mysql_stmt_close()</code>	Frees memory used by prepared statement.
<code>mysql_stmt_data_seek()</code>	Seeks to an arbitrary row number in a statement result set.
<code>mysql_stmt_errno()</code>	Returns the error number for the last statement execution.
<code>mysql_stmt_error()</code>	Returns the error message for the last statement execution.
<code>mysql_stmt_execute()</code>	Executes the prepared statement.
<code>mysql_stmt_fetch()</code>	Fetches the next row of data from the result set and returns data for all bound columns.
<code>mysql_stmt_fetch_column()</code>	Fetch data for one column of the current row of the result set.
<code>mysql_stmt_field_count()</code>	Returns the number of result columns for the most recent statement.
<code>mysql_stmt_free_result()</code>	Free the resources allocated to the statement handle.
<code>mysql_stmt_init()</code>	Allocates memory for <code>MYSQL_STMT</code> structure and initializes it.
<code>mysql_stmt_insert_id()</code>	Returns the ID generated for an <code>AUTO_INCREMENT</code> column by prepared statement.
<code>mysql_stmt_num_rows()</code>	Returns total rows from the statement buffered result set.
<code>mysql_stmt_param_count()</code>	Returns the number of parameters in a prepared SQL statement.
<code>mysql_stmt_param_metadata()</code>	(Return parameter metadata in the form of a result set.) Currently, this function does nothing.
<code>mysql_stmt_prepare()</code>	Prepares an SQL string for execution.
<code>mysql_stmt_reset()</code>	Reset the statement buffers in the server.
<code>mysql_stmt_result_metadata()</code>	Returns prepared statement metadata in the form of a result set.
<code>mysql_stmt_row_seek()</code>	Seeks to a row offset in a statement result set, using value returned from <code>mysql_stmt_row_tell()</code> .
<code>mysql_stmt_row_tell()</code>	Returns the statement row cursor position.
<code>mysql_stmt_send_long_data()</code>	Sends long data in chunks to server.
<code>mysql_stmt_sqlstate()</code>	Returns the SQLSTATE error code for the last statement execution.
<code>mysql_stmt_store_result()</code>	Retrieves the complete result set to the client.

Call `mysql_stmt_init()` to create a statement handle, then `mysql_stmt_prepare` to prepare it, `mysql_stmt_bind_param()` to supply the parameter data, and `mysql_stmt_execute()` to execute the statement. You can repeat the `mysql_stmt_execute()` by changing parameter values in the respective buffers supplied through

`mysql_stmt_bind_param()`.

If the statement is a `SELECT` or any other statement that produces a result set, `mysql_stmt_prepare()` also returns the result set metadata information in the form of a `MYSQL_RES` result set through `mysql_stmt_result_metadata()`.

You can supply the result buffers using `mysql_stmt_bind_result()`, so that the `mysql_stmt_fetch()` automatically returns data to these buffers. This is row-by-row fetching.

You can also send the text or binary data in chunks to server using `mysql_stmt_send_long_data()`. See [Section 25.2.7.25](#), “`mysql_stmt_send_long_data()`”.

When statement execution has been completed, the statement handle must be closed using `mysql_stmt_close()` so that all resources associated with it can be freed.

If you obtained a `SELECT` statement's result set metadata by calling `mysql_stmt_result_metadata()`, you should also free the metadata using `mysql_free_result()`.

Execution Steps

To prepare and execute a statement, an application follows these steps:

1. Create a prepared statement handle with `mysql_stmt_init()`. To prepare the statement on the server, call `mysql_stmt_prepare()` and pass it a string containing the SQL statement.
2. If the statement produces a result set, call `mysql_stmt_result_metadata()` to obtain the result set metadata. This metadata is itself in the form of result set, albeit a separate one from the one that contains the rows returned by the query. The metadata result set indicates how many columns are in the result and contains information about each column.
3. Set the values of any parameters using `mysql_stmt_bind_param()`. All parameters must be set. Otherwise, statement execution returns an error or produces unexpected results.
4. Call `mysql_stmt_execute()` to execute the statement.
5. If the statement produces a result set, bind the data buffers to use for retrieving the row values by calling `mysql_stmt_bind_result()`.
6. Fetch the data into the buffers row by row by calling `mysql_stmt_fetch()` repeatedly until no more rows are found.
7. Repeat steps 3 through 6 as necessary, by changing the parameter values and re-executing the statement.

When `mysql_stmt_prepare()` is called, the MySQL client/server protocol performs these actions:

- The server parses the statement and sends the okay status back to the client by assigning a statement ID. It also sends total number of parameters, a column count, and its metadata if it is a result set oriented statement. All syntax and semantics of the statement are checked by the server during this call.
- The client uses this statement ID for the further operations, so that the server can identify the statement from among its pool of statements.

When `mysql_stmt_execute()` is called, the MySQL client/server protocol performs these actions:

- The client uses the statement handle and sends the parameter data to the server.
- The server identifies the statement using the ID provided by the client, replaces the parameter markers with the newly supplied data, and executes the statement. If the statement produces a result set, the server sends the data back to the client. Otherwise, it sends an okay status and total number of rows changed, deleted, or inserted.

When `mysql_stmt_fetch()` is called, the MySQL client/server protocol performs these actions:

- The client reads the data from the packet row by row and places it into the application data buffers by doing the necessary conversions. If the application buffer type is same as that of the field type returned from the server, the conversions are straightforward.

If an error occurs, you can get the statement error code, error message, and SQLSTATE value using `mysql_stmt_errno()`, `mysql_stmt_error()`, and `mysql_stmt_sqlstate()`, respectively.

Prepared Statement Logging

For prepared statements that are executed with the `mysql_stmt_prepare()` and `mysql_stmt_execute()` C API functions, the server writes **Prepare** and **Execute** lines to the general query log so that you can tell when statements are prepared and executed.

Suppose that you prepare and execute a statement as follows:

1. Call `mysql_stmt_prepare()` to prepare the statement string `"SELECT ?"`.
2. Call `mysql_stmt_bind_param()` to bind the value `3` to the parameter in the prepared statement.
3. Call `mysql_stmt_execute()` to execute the prepared statement.

As a result of the preceding calls, the server writes the following lines to the general query log:

```
Prepare [1] SELECT ?
Execute [1] SELECT 3
```

Each **Prepare** and **Execute** line in the log is tagged with a `[N]` statement identifier so that you can keep track of which prepared statement is being logged. `N` is a positive integer. If there are multiple prepared statements active simultaneously for the client, `N` may be greater than 1. Each **Execute** line shows a prepared statement after substitution of data values for `?` parameters.

Version notes: **Prepare** lines are displayed without `[N]` before MySQL 4.1.10. **Execute** lines are not displayed at all before MySQL 4.1.10.

25.2.7. C API Prepared Statement Function Descriptions

To prepare and execute queries, use the functions described in detail in the following sections.

Note that all functions operating with a `MYSQL_STMT` structure begin with the prefix `mysql_stmt_`.

To create a `MYSQL_STMT` handle, use the `mysql_stmt_init()` function.

25.2.7.1. `mysql_stmt_affected_rows()`

```
my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)
```

Description

Returns the total number of rows changed, deleted, or inserted by the last executed statement. May be called immediately after `mysql_stmt_execute()` for **UPDATE**, **DELETE**, or **INSERT** statements. For **SELECT** statements, `mysql_stmt_affected_rows()` works like `mysql_num_rows()`.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an **UPDATE** statement, no rows matched the **WHERE** clause in the query, or that no query has yet been executed. -1 indicates that the query returned an error or that, for a **SELECT** query, `mysql_stmt_affected_rows()` was called prior to calling `mysql_stmt_store_result()`. Because `mysql_stmt_affected_rows()` returns an unsigned value, you can check for -1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent).

See [Section 25.2.3.1](#), “`mysql_affected_rows()`”, for additional information on the return value.

Errors

None.

Example

For the usage of `mysql_stmt_affected_rows()`, refer to the Example from [Section 25.2.7.10](#), “`mysql_stmt_execute()`”.

25.2.7.2. `mysql_stmt_attr_get()`

```
int mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type option, void *arg)
```

Description

Can be used to get the current value for a statement attribute.

The `option` argument is the option that you want to get; the `arg` should point to a variable that should contain the option value. If the option is an integer, then `arg` should point to the value of the integer.

See [Section 25.2.7.3](#), “`mysql_stmt_attr_set()`”, for a list of options and option types.

Note

In MySQL 5.1, `mysql_stmt_attr_get()` originally used `unsigned int *`, not `my_bool *`, for `STMT_ATTR_UPDATE_MAX_LENGTH`. This was corrected in MySQL 5.1.7.

Return Values

Zero if successful. Non-zero if `option` is unknown.

Errors

None.

25.2.7.3. `mysql_stmt_attr_set()`

```
int mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option, const void *arg)
```

Description

Can be used to affect behavior for a prepared statement. This function may be called multiple times to set several options.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, then `arg` should point to the value of the integer.

Possible `option` values:

Option	Argument Type	Function
<code>STMT_ATTR_UPDATE_MAX_LENGTH</code>	<code>my_bool *</code>	If set to 1: Update metadata <code>MYSQL_FIELD->max_length</code> in <code>mysql_stmt_store_result()</code> .
<code>STMT_ATTR_CURSOR_TYPE</code>	<code>unsigned long *</code>	Type of cursor to open for statement when <code>mysql_stmt_execute()</code> is invoked. <code>*arg</code> can be <code>CURSOR_TYPE_NO_CURSOR</code> (the default) or <code>CURSOR_TYPE_READ_ONLY</code> .
<code>STMT_ATTR_PREFETCH_ROWS</code>	<code>unsigned long *</code>	Number of rows to fetch from server at a time when using a cursor. <code>*arg</code> can be in the range from 1 to the maximum value of <code>unsigned long</code> . The default is 1.

If you use the `STMT_ATTR_CURSOR_TYPE` option with `CURSOR_TYPE_READ_ONLY`, a cursor is opened for the statement when you invoke `mysql_stmt_execute()`. If there is already an open cursor from a previous `mysql_stmt_execute()` call, it closes the cursor before opening a new one. `mysql_stmt_reset()` also closes any open cursor before preparing the statement for re-execution. `mysql_stmt_free_result()` closes any open cursor.

If you open a cursor for a prepared statement, `mysql_stmt_store_result()` is unnecessary, because that function causes the result set to be buffered on the client side.

Return Values

Zero if successful. Non-zero if `option` is unknown.

Errors

None.

Example

The following example opens a cursor for a prepared statement and sets the number of rows to fetch at a time to 5:

```
MYSQL_STMT *stmt;
int rc;
unsigned long type;
unsigned long prefetch_rows = 5;

stmt = mysql_stmt_init(mysql);
type = (unsigned long) CURSOR_TYPE_READ_ONLY;
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_CURSOR_TYPE, (void*) &type);
/* ... check return value ... */
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_PREFETCH_ROWS,
                        (void*) &prefetch_rows);
/* ... check return value ... */
```

25.2.7.4. `mysql_stmt_bind_param()`

```
my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_param()` is used to bind input data for the parameter markers in the SQL statement that was passed to `mysql_stmt_prepare()`. It uses `MYSQL_BIND` structures to supply the data. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain one element for each ‘?’ parameter marker that is present in the query.

Suppose that you prepare the following statement:

```
INSERT INTO mytbl VALUES(?,?,?)
```

When you bind the parameters, the array of `MYSQL_BIND` structures must contain three elements, and can be declared like this:

```
MYSQL_BIND bind[3];
```

[Section 25.2.5, “C API Prepared Statement Data types”](#), describes the members of each `MYSQL_BIND` element and how they should be set to provide input values.

Return Values

Zero if the bind operation was successful. Non-zero if an error occurred.

Errors

- `CR_UNSUPPORTED_PARAM_TYPE`

The conversion is not supported. Possibly the `buffer_type` value is illegal or is not one of the supported types.

- `CR_OUT_OF_MEMORY`

Out of memory.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

Example

For the usage of `mysql_stmt_bind_param()`, refer to the Example from [Section 25.2.7.10](#), “`mysql_stmt_execute()`”.

25.2.7.5. `mysql_stmt_bind_result()`

```
my_bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_result()` is used to associate (that is, bind) output columns in the result set to data buffers and length buffers. When `mysql_stmt_fetch()` is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified buffers.

All columns must be bound to buffers prior to calling `mysql_stmt_fetch()`. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain one element for each column of the result set. If you do not bind columns to `MYSQL_BIND` structures, `mysql_stmt_fetch()` simply ignores the data fetch. The buffers should be large enough to hold the data values, because the protocol doesn't return data values in chunks.

A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time `mysql_stmt_fetch()` is called. Suppose that an application binds the columns in a result set and calls `mysql_stmt_fetch()`. The client/server protocol returns data in the bound buffers. Then suppose that the application binds the columns to a different set of buffers. The protocol places data into the newly bound buffers when the next call to `mysql_stmt_fetch()` occurs.

To bind a column, an application calls `mysql_stmt_bind_result()` and passes the type, address, and length of the output buffer into which the value should be stored. [Section 25.2.5](#), “C API Prepared Statement Data types”, describes the members of each `MYSQL_BIND` element and how they should be set to receive output values.

Return Values

Zero if the bind operation was successful. Non-zero if an error occurred.

Errors

- `CR_UNSUPPORTED_PARAM_TYPE`
The conversion is not supported. Possibly the `buffer_type` value is illegal or is not one of the supported types.
- `CR_OUT_OF_MEMORY`
Out of memory.
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

Example

For the usage of `mysql_stmt_bind_result()`, refer to the Example from [Section 25.2.7.11](#), “`mysql_stmt_fetch()`”.

25.2.7.6. `mysql_stmt_close()`

```
my_bool mysql_stmt_close(MYSQL_STMT *)
```

Description

Closes the prepared statement. `mysql_stmt_close()` also deallocates the statement handle pointed to by `stmt`.

If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

Return Values

Zero if the statement was freed successfully. Non-zero if an error occurred.

Errors

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

Example

For the usage of `mysql_stmt_close()`, refer to the Example from [Section 25.2.7.10](#), “`mysql_stmt_execute()`”.

25.2.7.7. `mysql_stmt_data_seek()`

```
void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a statement result set. The `offset` value is a row number and should be in the range from 0 to `mysql_stmt_num_rows(stmt)-1`.

This function requires that the statement result set structure contains the entire result of the last executed query, so `mysql_stmt_data_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

None.

Errors

None.

25.2.7.8. `mysql_stmt_errno()`

```
unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_errno()` returns the error code for the most recently invoked statement API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at [Appendix B](#), *Errors, Error Codes, and Common Problems*.

Return Values

An error code value. Zero if no error occurred.

Errors

None.

25.2.7.9. `mysql_stmt_error()`

```
const char *mysql_stmt_error(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_error()` returns a null-terminated string containing the error message for the most recently invoked statement API function that can succeed or fail. An empty string (“”) is returned if no error occurred. This means the following two tests are equivalent:

```
if (mysql_stmt_errno(stmt))
{
    // an error occurred
}
```



```
if (mysql_stmt_error(stmt)[0])
{
    // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently, you can choose error messages in several different languages.

Return Values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

25.2.7.10. `mysql_stmt_execute()`

```
int mysql_stmt_execute(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_execute()` executes the prepared query associated with the statement handle. The currently bound parameter marker values are sent to server during this call, and the server replaces the markers with this newly supplied data.

If the statement is an `UPDATE`, `DELETE`, or `INSERT`, the total number of changed, deleted, or inserted rows can be found by calling `mysql_stmt_affected_rows()`. If this is a statement such as `SELECT` that generates a result set, you must call `mysql_stmt_fetch()` to fetch the data prior to calling any other functions that result in query processing. For more information on how to fetch the results, refer to [Section 25.2.7.11, “mysql_stmt_fetch\(\)”](#).

For statements that generate a result set, you can request that `mysql_stmt_execute()` open a cursor for the statement by calling `mysql_stmt_attr_set()` before executing the statement. If you execute a statement multiple times, `mysql_stmt_execute()` closes any open cursor before opening a new one.

Return Values

Zero if execution was successful. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_OUT_OF_MEMORY`
Out of memory.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`
The connection to the server was lost during the query.
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

Example

The following example demonstrates how to create and populate a table using `mysql_stmt_init()`, `mysql_stmt_prepare()`, `mysql_stmt_param_count()`, `mysql_stmt_bind_param()`, `mysql_stmt_execute()`, and `mysql_stmt_affected_rows()`. The `mysql` variable is assumed to be a valid connection handle.

```

#define STRING_SIZE 50

#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(col1 INT,\
                                col2 VARCHAR(40),\
                                col3 SMALLINT,\
                                col4 TIMESTAMP)"

#define INSERT_SAMPLE "INSERT INTO \
    test_table(col1,col2,col3) \
    VALUES(?,?,?)"

MYSQL_STMT      *stmt;
MYSQL_BIND      bind[3];
my_ulonglong    affected_rows;
int             param_count;
short           small_data;
int             int_data;
char            str_data[STRING_SIZE];
unsigned long   str_length;
my_bool         is_null;

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
    fprintf(stderr, " DROP TABLE failed\n");
    fprintf(stderr, " %s\n", mysql_error(mysql));
    exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
    fprintf(stderr, " CREATE TABLE failed\n");
    fprintf(stderr, " %s\n", mysql_error(mysql));
    exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; the server */
/* sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
    fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
param_count = mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
    fprintf(stderr, " invalid parameter count returned by MySQL\n");
    exit(0);
}

/* Bind the data for all 3 parameters */
memset(bind, 0, sizeof(bind));

/* INTEGER PARAM */
/* This is a number type, so there is no need
   to specify buffer_length */
bind[0].buffer_type = MYSQL_TYPE_LONG;
bind[0].buffer = (char *)&int_data;
bind[0].is_null = 0;
bind[0].length = 0;

/* STRING PARAM */
bind[1].buffer_type = MYSQL_TYPE_STRING;
bind[1].buffer = (char *)str_data;
bind[1].buffer_length = STRING_SIZE;
bind[1].is_null = 0;
bind[1].length = &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type = MYSQL_TYPE_SHORT;
bind[2].buffer = (char *)&small_data;
bind[2].is_null = &is_null;
bind[2].length = 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{

```

```

    fprintf(stderr, " mysql_stmt_bind_param() failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Specify the data values for the first row */
int_data= 10; /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get the total number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %lu\n",
        (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
    fprintf(stderr, " invalid affected rows by MySQL\n");
    exit(0);
}

/* Specify data values for second row,
   then re-execute the statement */
int_data= 1000;
strncpy(str_data, "
    The most popular Open Source database",
        STRING_SIZE);
str_length= strlen(str_data);
small_data= 1000; /* smallint */
is_null= 0; /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, " mysql_stmt_execute, 2 failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 2): %lu\n",
        (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
    fprintf(stderr, " invalid affected rows by MySQL\n");
    exit(0);
}

/* Close the statement */
if (mysql_stmt_close(stmt))
{
    fprintf(stderr, " failed while closing the statement\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

```

Note

For complete examples on the use of prepared statement functions, refer to the file [tests/mysql_client_test.c](#). This file can be obtained from a MySQL source distribution or from the BitKeeper source repository.

25.2.7.11. `mysql_stmt_fetch()`

```
int mysql_stmt_fetch(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_fetch()` returns the next row in the result set. It can be called only while the result set exists; that is, after a call to `mysql_stmt_execute()` for a statement such as `SELECT` that creates a result set.

`mysql_stmt_fetch()` returns row data using the buffers bound by `mysql_stmt_bind_result()`. It returns the data in those buffers for all the columns in the current row set and the lengths are returned to the `length` pointer. All columns must be bound by the application before it calls `mysql_stmt_fetch()`.

By default, result sets are fetched unbuffered a row at a time from the server. To buffer the entire result set on the client, call `mysql_stmt_store_result()` after binding the data buffers and before calling `mysql_stmt_fetch()`.

If a fetched data value is a `NULL` value, the `*is_null` value of the corresponding `MYSQL_BIND` structure contains `TRUE` (1). Otherwise, the data and its length are returned in the `*buffer` and `*length` elements based on the buffer type specified by the application. Each numeric and temporal type has a fixed length, as listed in the following table. The length of the string types depends on the length of the actual data value, as indicated by `data_length`.

Type	Length
<code>MYSQL_TYPE_TINY</code>	1
<code>MYSQL_TYPE_SHORT</code>	2
<code>MYSQL_TYPE_LONG</code>	4
<code>MYSQL_TYPE_LONGLONG</code>	8
<code>MYSQL_TYPE_FLOAT</code>	4
<code>MYSQL_TYPE_DOUBLE</code>	8
<code>MYSQL_TYPE_TIME</code>	<code>sizeof(MYSQL_TIME)</code>
<code>MYSQL_TYPE_DATE</code>	<code>sizeof(MYSQL_TIME)</code>
<code>MYSQL_TYPE_DATETIME</code>	<code>sizeof(MYSQL_TIME)</code>
<code>MYSQL_TYPE_STRING</code>	<code>data_length</code>
<code>MYSQL_TYPE_BLOB</code>	<code>data_length</code>

Return Values

Return Value	Description
0	Successful, the data has been fetched to application data buffers.
1	Error occurred. Error code and message can be obtained by calling <code>mysql_stmt_errno()</code> and <code>mysql_stmt_error()</code> .
<code>MYSQL_NO_DATA</code>	No more rows/data exists
<code>MYSQL_DATA_TRUNCATED</code>	Data truncation occurred

`MYSQL_DATA_TRUNCATED` is returned when truncation reporting is enabled. (Reporting is enabled by default, but can be controlled with `mysql_options()`.) To determine which parameters were truncated when this value is returned, check the `error` members of the `MYSQL_BIND` parameter structures.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_OUT_OF_MEMORY`
Out of memory.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`
The connection to the server was lost during the query.

- [CR_UNKNOWN_ERROR](#)

An unknown error occurred.

- [CR_UNSUPPORTED_PARAM_TYPE](#)

The buffer type is [MYSQL_TYPE_DATE](#), [MYSQL_TYPE_TIME](#), [MYSQL_TYPE_DATETIME](#), or [MYSQL_TYPE_TIMESTAMP](#), but the data type is not [DATE](#), [TIME](#), [DATETIME](#), or [TIMESTAMP](#).

- All other unsupported conversion errors are returned from [mysql_stmt_bind_result\(\)](#).

Example

The following example demonstrates how to fetch data from a table using [mysql_stmt_result_metadata\(\)](#), [mysql_stmt_bind_result\(\)](#), and [mysql_stmt_fetch\(\)](#). (This example expects to retrieve the two rows inserted by the example shown in [Section 25.2.7.10](#), “[mysql_stmt_execute\(\)](#)”.) The [mysql](#) variable is assumed to be a valid connection handle.

```
#define STRING_SIZE 50

#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 \
                      FROM test_table"

MYSQL_STMT      *stmt;
MYSQL_BIND      bind[4];
MYSQL_RES       *prepare_meta_result;
MYSQL_TIME      ts;
unsigned long    length[4];
int              param_count, column_count, row_count;
short            small_data;
int              int_data;
char             str_data[STRING_SIZE];
my_bool          is_null[4];
my_bool          error[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
    fprintf(stderr, " mysql_stmt_prepare(), SELECT failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count = mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
{
    fprintf(stderr, " invalid parameter count returned by MySQL\n");
    exit(0);
}

/* Fetch result set meta information */
prepare_meta_result = mysql_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
    fprintf(stderr,
            " mysql_stmt_result_metadata(), \
            returned no meta information\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get total columns in the query */
column_count = mysql_num_fields(prepare_meta_result);
fprintf(stdout,
        " total columns in SELECT statement: %d\n",
        column_count);

if (column_count != 4) /* validate column count */
{
    fprintf(stderr, " invalid column count returned by MySQL\n");
    exit(0);
}
```

```

/* Execute the SELECT query */
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, " mysql_stmt_execute(), failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */
memset(bind, 0, sizeof(bind));

/* INTEGER COLUMN */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];
bind[0].error= &error[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)&str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];
bind[1].error= &error[1];

/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];
bind[2].error= &error[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
bind[3].length= &length[3];
bind[3].error= &error[3];

/* Bind the result buffers */
if (mysql_stmt_bind_result(stmt, bind))
{
    fprintf(stderr, " mysql_stmt_bind_result() failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Now buffer all results to client (optional step) */
if (mysql_stmt_store_result(stmt))
{
    fprintf(stderr, " mysql_stmt_store_result() failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ...\n");
while (!mysql_stmt_fetch(stmt))
{
    row_count++;
    fprintf(stdout, " row %d\n", row_count);

    /* column 1 */
    fprintf(stdout, " column1 (integer) : ");
    if (is_null[0])
        fprintf(stdout, " NULL\n");
    else
        fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

    /* column 2 */
    fprintf(stdout, " column2 (string) : ");
    if (is_null[1])
        fprintf(stdout, " NULL\n");
    else
        fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

    /* column 3 */
    fprintf(stdout, " column3 (smallint) : ");
    if (is_null[2])
        fprintf(stdout, " NULL\n");
    else
        fprintf(stdout, " %d(%ld)\n", small_data, length[2]);

    /* column 4 */
    fprintf(stdout, " column4 (timestamp): ");
    if (is_null[3])
        fprintf(stdout, " NULL\n");

```

```

    else
        fprintf(stdout, " %04d-%02d-%02d %02d:%02d:%02d (%ld)\n",
                    ts.year, ts.month, ts.day,
                    ts.hour, ts.minute, ts.second,
                    length[3]);
    fprintf(stdout, "\n");
}

/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
    fprintf(stderr, " MySQL failed to return all rows\n");
    exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);

/* Close the statement */
if (mysql_stmt_close(stmt))
{
    fprintf(stderr, " failed while closing the statement\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

```

In some cases you might want to determine the length of a column value before fetching it with `mysql_stmt_fetch()`. For example, the value might be a long string or **BLOB** value for which you want to know how much space must be allocated. To accomplish this, you can use these strategies:

- Before invoking `mysql_stmt_fetch()` to retrieve individual rows, invoke `mysql_stmt_store_result()` to buffer the entire result on the client side. Then the maximal length of column values will be indicated by the `max_length` member of the result set metadata returned by `mysql_stmt_result_metadata()`. This strategy requires that you pass `STMT_ATTR_UPDATE_MAX_LENGTH` to `mysql_stmt_attr_set()` or the `max_length` values will not be calculated.
- Invoke `mysql_stmt_fetch()` with a zero-length buffer for the column in question and a pointer in which the real length can be stored. Then use the real length with `mysql_stmt_fetch_column()`.

```

real_length= 0;

bind[0].buffer= 0;
bind[0].buffer_length= 0;
bind[0].length= &real_length
mysql_stmt_bind_result(stmt, bind);

mysql_stmt_fetch(stmt);
if (real_length > 0)
{
    data= malloc(real_length);
    bind[0].buffer= data;
    bind[0].buffer_length= real_length;
    mysql_stmt_fetch_column(stmt, 0, bind, 0);
}

```

25.2.7.12. `mysql_stmt_fetch_column()`

```
int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, unsigned int column, unsigned long offset)
```

Description

Fetch one column from the current result set row. `bind` provides the buffer where data should be placed. It should be set up the same way as for `mysql_stmt_bind_result()`. `column` indicates which column to fetch. The first column is numbered 0. `offset` is the offset within the data value at which to begin retrieving data. This can be used for fetching the data value in pieces. The beginning of the value is offset 0.

Return Values

Zero if the value was fetched successfully. Non-zero if an error occurred.

Errors

- `CR_INVALID_PARAMETER_NO`

Invalid column number.

- `CR_NO_DATA`

The end of the result set has already been reached.

25.2.7.13. `mysql_stmt_field_count()`

```
unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)
```

Description

Returns the number of columns for the most recent statement for the statement handler. This value is zero for statements such as `INSERT` or `DELETE` that do not produce result sets.

`mysql_stmt_field_count()` can be called after you have prepared a statement by invoking `mysql_stmt_prepare()`.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

25.2.7.14. `mysql_stmt_free_result()`

```
my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)
```

Description

Releases memory associated with the result set produced by execution of the prepared statement. If there is a cursor open for the statement, `mysql_stmt_free_result()` closes it.

Return Values

Zero if the result set was freed successfully. Non-zero if an error occurred.

Errors

25.2.7.15. `mysql_stmt_init()`

```
MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)
```

Description

Create a `MYSQL_STMT` handle. The handle should be freed with `mysql_stmt_close(MYSQL_STMT *)`.

Return values

A pointer to a `MYSQL_STMT` structure in case of success. `NULL` if out of memory.

Errors

- `CR_OUT_OF_MEMORY`

Out of memory.

25.2.7.16. `mysql_stmt_insert_id()`

```
my_ulonglong mysql_stmt_insert_id(MYSQL_STMT *stmt)
```


Description

Returns the value generated for an [AUTO_INCREMENT](#) column by the prepared [INSERT](#) or [UPDATE](#) statement. Use this function after you have executed a prepared [INSERT](#) statement on a table which contains an [AUTO_INCREMENT](#) field.

See [Section 25.2.3.37](#), “[mysql_insert_id\(\)](#)”, for more information.

Return Values

Value for [AUTO_INCREMENT](#) column which was automatically generated or explicitly set during execution of prepared statement, or value generated by [LAST_INSERT_ID\(*expr*\)](#) function. Return value is undefined if statement does not set [AUTO_INCREMENT](#) value.

Errors

None.

25.2.7.17. [mysql_stmt_num_rows\(\)](#)

```
my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)
```

Description

Returns the number of rows in the result set.

The use of [mysql_stmt_num_rows\(\)](#) depends on whether you used [mysql_stmt_store_result\(\)](#) to buffer the entire result set in the statement handle.

If you use [mysql_stmt_store_result\(\)](#), [mysql_stmt_num_rows\(\)](#) may be called immediately.

[mysql_stmt_num_rows\(\)](#) is intended for use with statements that return a result set, such as [SELECT](#). For statements such as [INSERT](#), [UPDATE](#), or [DELETE](#), the number of affected rows can be obtained with [mysql_stmt_affected_rows\(\)](#).

Return Values

The number of rows in the result set.

Errors

None.

25.2.7.18. [mysql_stmt_param_count\(\)](#)

```
unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)
```

Description

Returns the number of parameter markers present in the prepared statement.

Return Values

An unsigned long integer representing the number of parameters in a statement.

Errors

None.

Example

For the usage of [mysql_stmt_param_count\(\)](#), refer to the Example from [Section 25.2.7.10](#), “[mysql_stmt_execute\(\)](#)”.

25.2.7.19. [mysql_stmt_param_metadata\(\)](#)

```
MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)
```

This function currently does nothing.

Description**Return Values****Errors****25.2.7.20. `mysql_stmt_prepare()`**

```
int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *stmt_str, unsigned long length)
```

Description

Given the statement handle returned by `mysql_stmt_init()`, prepares the SQL statement pointed to by the string `stmt_str` and returns a status value. The string length should be given by the `length` argument. The string must consist of a single SQL statement. You should not add a terminating semicolon (`;`) or `\g` to the statement.

The application can include one or more parameter markers in the SQL statement by embedding question mark (`'?`') characters into the SQL string at the appropriate positions.

The markers are legal only in certain places in SQL statements. For example, they are allowed in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value. However, they are not allowed for identifiers (such as table or column names), or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

The parameter markers must be bound to application variables using `mysql_stmt_bind_param()` before executing the statement.

Return Values

Zero if the statement was prepared successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_OUT_OF_MEMORY`
Out of memory.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`
The connection to the server was lost during the query
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

If the prepare operation was unsuccessful (that is, `mysql_stmt_prepare()` returns non-zero), the error message can be obtained by calling `mysql_stmt_error()`.

Example

For the usage of `mysql_stmt_prepare()`, refer to the Example from [Section 25.2.7.10, “`mysql\_stmt\_execute\(\)`”](#).

25.2.7.21. `mysql_stmt_reset()`

```
my_bool mysql_stmt_reset(MYSQL_STMT *stmt)
```

Description

Reset the prepared statement on the client and server to state after prepare. This is mainly used to reset data sent with `mysql_stmt_send_long_data()`. Any open cursor for the statement is closed.

To re-prepare the statement with another query, use `mysql_stmt_prepare()`.

Return Values

Zero if the statement was reset successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`
Commands were executed in an improper order.
- `CR_SERVER_GONE_ERROR`
The MySQL server has gone away.
- `CR_SERVER_LOST`
The connection to the server was lost during the query
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

25.2.7.22. `mysql_stmt_result_metadata()`

```
MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)
```

Description

If a statement passed to `mysql_stmt_prepare()` is one that produces a result set, `mysql_stmt_result_metadata()` returns the result set metadata in the form of a pointer to a `MYSQL_RES` structure that can be used to process the meta information such as total number of fields and individual field information. This result set pointer can be passed as an argument to any of the field-based API functions that process result set metadata, such as:

- `mysql_num_fields()`
- `mysql_fetch_field()`
- `mysql_fetch_field_direct()`
- `mysql_fetch_fields()`
- `mysql_field_count()`
- `mysql_field_seek()`
- `mysql_field_tell()`
- `mysql_free_result()`

The result set structure should be freed when you are done with it, which you can do by passing it to `mysql_free_result()`. This is similar to the way you free a result set obtained from a call to `mysql_store_result()`.

The result set returned by `mysql_stmt_result_metadata()` contains only metadata. It does not contain any row results. The rows are obtained by using the statement handle with `mysql_stmt_fetch()`.

Return Values

A `MYSQL_RES` result structure. `NULL` if no meta information exists for the prepared query.

Errors

- `CR_OUT_OF_MEMORY`
Out of memory.
- `CR_UNKNOWN_ERROR`
An unknown error occurred.

Example

For the usage of `mysql_stmt_result_metadata()`, refer to the Example from [Section 25.2.7.11](#), “`mysql_stmt_fetch()`”.

25.2.7.23. `mysql_stmt_row_seek()`

```
MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a statement result set. The `offset` value is a row offset that should be a value returned from `mysql_stmt_row_tell()` or from `mysql_stmt_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_stmt_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_stmt_row_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_stmt_row_seek()`.

Errors

None.

25.2.7.24. `mysql_stmt_row_tell()`

```
MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)
```

Description

Returns the current position of the row cursor for the last `mysql_stmt_fetch()`. This value can be used as an argument to `mysql_stmt_row_seek()`.

You should use `mysql_stmt_row_tell()` only after `mysql_stmt_store_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

25.2.7.25. `mysql_stmt_send_long_data()`

```
my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int parameter_number, const char *data, unsigned long length)
```

Description

Allows an application to send parameter data to the server in pieces (or “chunks”). Call this function after

`mysql_stmt_bind_param()` and before `mysql_stmt_execute()`. It can be called multiple times to send the parts of a character or binary data value for a column, which must be one of the **TEXT** or **BLOB** data types.

`parameter_number` indicates which parameter to associate the data with. Parameters are numbered beginning with 0. `data` is a pointer to a buffer containing data to be sent, and `length` indicates the number of bytes in the buffer.

Note

The next `mysql_stmt_execute()` call ignores the bind buffer for all parameters that have been used with `mysql_stmt_send_long_data()` since last `mysql_stmt_execute()` or `mysql_stmt_reset()`.

If you want to reset/forget the sent data, you can do it with `mysql_stmt_reset()`. See [Section 25.2.7.21](#), “`mysql_stmt_reset()`”.

Return Values

Zero if the data is sent successfully to server. Non-zero if an error occurred.

Errors

- **CR_INVALID_BUFFER_USE**
The parameter does not have a string or binary type.
- **CR_COMMANDS_OUT_OF_SYNC**
Commands were executed in an improper order.
- **CR_SERVER_GONE_ERROR**
The MySQL server has gone away.
- **CR_OUT_OF_MEMORY**
Out of memory.
- **CR_UNKNOWN_ERROR**
An unknown error occurred.

Example

The following example demonstrates how to send the data for a **TEXT** column in chunks. It inserts the data value 'MySQL - The most popular Open Source database' into the `text_column` column. The `mysql` variable is assumed to be a valid connection handle.

```
#define INSERT_QUERY "INSERT INTO \
    test_long_data(text_column) VALUES(?)"

MYSQL_BIND bind[1];
long          length;

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
{
    fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}
memset(bind, 0, sizeof(bind));
bind[0].buffer_type= MYSQL_TYPE_STRING;
bind[0].length= &length;
bind[0].is_null= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
```

```

fprintf(stderr, "\n param bind failed");
fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
exit(0);
}

/* Supply data in chunks to server */
if (mysql_stmt_send_long_data(stmt,0,"MySQL",5))
{
fprintf(stderr, "\n send_long_data failed");
fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
exit(0);
}

/* Supply the next piece of data */
if (mysql_stmt_send_long_data(stmt,0,
    " - The most popular Open Source database",40))
{
fprintf(stderr, "\n send_long_data failed");
fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
exit(0);
}

/* Now, execute the query */
if (mysql_stmt_execute(stmt))
{
fprintf(stderr, "\n mysql_stmt_execute failed");
fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
exit(0);
}

```

25.2.7.26. `mysql_stmt_sqlstate()`

```
const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_sqlstate()` returns a null-terminated string containing the SQLSTATE error code for the most recently invoked prepared statement API function that can succeed or fail. The error code consists of five characters. "00000" means "no error." The values are specified by ANSI SQL and ODBC. For a list of possible values, see [Appendix B, Errors, Error Codes, and Common Problems](#).

Note that not all MySQL errors are yet mapped to SQLSTATE codes. The value "HY000" (general error) is used for unmapped errors.

Return Values

A null-terminated character string containing the SQLSTATE error code.

25.2.7.27. `mysql_stmt_store_result()`

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```

Description

Result sets are produced by executing prepared statements for SQL statements such as [SELECT](#), [SHOW](#), [DESCRIBE](#), and [EXPLAIN](#). By default, result sets for successfully executed prepared statements are not buffered on the client and `mysql_stmt_fetch()` fetches them one at a time from the server. To cause the complete result set to be buffered on the client, call `mysql_stmt_store_result()` after binding data buffers with `mysql_stmt_bind_result()` and before calling `mysql_stmt_fetch()` to fetch rows. (For an example, see [Section 25.2.7.11, "mysql_stmt_fetch\(\)"](#).)

`mysql_stmt_store_result()` is optional for result set processing, unless you will call `mysql_stmt_data_seek()`, `mysql_stmt_row_seek()`, or `mysql_stmt_row_tell()`. Those functions require a seekable result set.

It is unnecessary to call `mysql_stmt_store_result()` after executing a SQL statement that does not produce a result set, but if you do, it does not harm or cause any notable performance problem. You can detect whether the statement produced a result set by checking if `mysql_stmt_result_metadata()` returns `NULL`. For more information, refer to [Section 25.2.7.22, "mysql_stmt_result_metadata\(\)"](#).

Note

MySQL doesn't by default calculate `MYSQL_FIELD->max_length` for all columns in `mysql_stmt_store_result()` because calculating this would slow down `mysql_stmt_store_result()` considerably and most applications doesn't need `max_length`. If you want `max_length` to be updated, you can call

`mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)` to enable this. See [Section 25.2.7.3](#), “`mysql_stmt_attr_set()`”.

Return Values

Zero if the results are buffered successfully. Non-zero if an error occurred.

Errors

- [CR_COMMANDS_OUT_OF_SYNC](#)
Commands were executed in an improper order.
- [CR_OUT_OF_MEMORY](#)
Out of memory.
- [CR_SERVER_GONE_ERROR](#)
The MySQL server has gone away.
- [CR_SERVER_LOST](#)
The connection to the server was lost during the query.
- [CR_UNKNOWN_ERROR](#)
An unknown error occurred.

25.2.8. C API Prepared statement problems

Here follows a list of the currently known problems with prepared statements:

- [TIME](#), [TIMESTAMP](#), and [DATETIME](#) do not support parts of seconds (for example from [DATE_FORMAT\(\)](#)).
- When converting an integer to string, [ZEROFILL](#) is honored with prepared statements in some cases where the MySQL server doesn't print the leading zeros. (For example, with [MIN\(*number-with-zerofill*\)](#)).
- When converting a floating point number to a string in the client, the rightmost digits of the converted value may differ slightly from those of the original value.
- Before MySQL 5.1.17, prepared statements do not use the query cache. As of 5.1.17, prepared statements use the query cache under the conditions described in [Section 6.5.4.1](#), “[How the Query Cache Operates](#)”.
- Prepared statements do not support multi-statements (that is, multiple statements within a single string separated by ‘;’ characters). This also means that prepared statements cannot invoke stored procedures that return result sets, because prepared statements do not support multiple result sets.

25.2.9. C API Handling of Multiple Statement Execution

By default, `mysql_query()` and `mysql_real_query()` interpret their statement string argument as a single statement to be executed, and you process the result according to whether the statement produces a result set (a set of rows, as for [SELECT](#)) or an affected-rows count (as for [INSERT](#), [UPDATE](#), and so forth).

MySQL 5.1 also supports the execution of a string containing multiple statements separated by semicolon (;) characters. This capability is enabled by special options that are specified either when you connect to the server with `mysql_real_connect()` or after connecting by calling `mysql_set_server_option()`.

Executing a multiple-statement string can produce multiple result sets or row-count indicators. Processing these results involves a different approach than for the single-statement case: After handling the result from the first statement, it is necessary to check whether more results exist and process them in turn if so. To support multiple-result processing, the C API includes the `mysql_more_results()` and `mysql_next_result()` functions. Generally, these functions are used at the end of a loop that

iterates as long as more results are available. *Failure to process the result this way may result in a dropped connection to the server.*

Multiple-result processing also is required if you execute `CALL` statements for stored procedures: A stored procedure returns a status result when it terminates, but it may also produce result sets as it runs (for example, if it executes `SELECT` statements). For any stored procedure that produces result sets in addition to the final status, you must be prepared to retrieve multiple results.

The multiple statement and result capabilities can be used only with `mysql_query()` or `mysql_real_query()`. They cannot be used with the prepared statement interface. Prepared statement handles are defined to work only with strings that contain a single statement. See [Section 25.2.4, “C API Prepared Statements”](#).

To enable multiple-statement execution and result processing, the following options may be used:

- The `mysql_real_connect()` function has a `flags` argument for which two option values are relevant:
 - `CLIENT_MULTI_RESULTS` enables the client program to process multiple results. This option *must* be enabled if you execute `CALL` statements for stored procedures that produce result sets. Otherwise, such procedures result in an error `Error 1312 (0A000): PROCEDURE proc_name can't return a result set in the given context`.
 - `CLIENT_MULTI_STATEMENTS` enables `mysql_query()` and `mysql_real_query()` to execute statement strings containing multiple statements separated by semicolons. This option also enables `CLIENT_MULTI_RESULTS` implicitly, so a `flags` argument of `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()` is equivalent to an argument of `CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS`. That is, `CLIENT_MULTI_STATEMENTS` is sufficient to enable multiple-statement execution and all multiple-result processing.
- After the connection to the server has been established, you can use the `mysql_set_server_option()` function to enable or disable multiple-statement execution by passing it an argument of `MYSQL_OPTION_MULTI_STATEMENTS_ON` or `MYSQL_OPTION_MULTI_STATEMENTS_OFF`. Enabling multiple-statement execution with this function also enables processing of “simple” results for a multiple-statement string where each statement produces a single result, but is *not* sufficient to allow processing of stored procedures that produce result sets.

The following procedure outlines a suggested strategy for handling multiple statements:

1. Pass `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()`, to fully enable multiple-statement execution and multiple-result processing.
2. After calling `mysql_query()` or `mysql_real_query()` and verifying that it succeeds, enter a loop within which you process statement results.
3. For each iteration of the loop, handle the current statement result, retrieving either a result set or an affected-rows count. If an error occurs, exit the loop.
4. At the end of the loop, call `mysql_next_result()` to check whether another result exists and initiate retrieval for it if so. If no more results are available, exit the loop.

One possible implementation of the preceding strategy is shown following.

```
/* connect to server with option CLIENT_MULTI_STATEMENTS */
if (mysql_real_connect (mysql, host_name, user_name, password,
    db_name, port_num, socket_name, CLIENT_MULTI_STATEMENTS) == NULL)
{
    printf("mysql_real_connect() failed\n");
    mysql_close(mysql);
    exit(1);
}

/* execute multiple statements */
status = mysql_query(mysql,
    "DROP TABLE IF EXISTS test_table;\n
    CREATE TABLE test_table(id INT);\n
    INSERT INTO test_table VALUES(10);\n
    UPDATE test_table SET id=20 WHERE id=10;\n
    SELECT * FROM test_table;\n
    DROP TABLE test_table");

if (status)
{
    printf("Could not execute statement(s)");
    mysql_close(mysql);
    exit(0);
}
```



```

/* process each statement result */
do {
    /* did current statement return data? */
    result = mysql_store_result(mysql);
    if (result)
    {
        /* yes; process rows and free the result set */
        process_result_set(mysql, result);
        mysql_free_result(result);
    }
    else /* no result set or error */
    {
        if (mysql_field_count(mysql) == 0)
        {
            printf("%lld rows affected\n",
                mysql_affected_rows(mysql));
        }
        else /* some error occurred */
        {
            printf("Could not retrieve result set\n");
            break;
        }
    }
} while ((status = mysql_next_result(mysql)) > 0);
printf("Could not execute statement\n");
mysql_close(mysql);

```

The final part of the loop can be reduced to a simple test of whether `mysql_next_result()` returns non-zero. The code as written distinguishes between no more results and an error, which allows a message to be printed for the latter occurrence.

25.2.10. C API Handling of Date and Time Values

The binary (prepared statement) protocol allows you to send and receive date and time values (`DATE`, `TIME`, `DATETIME`, and `TIMESTAMP`), using the `MYSQL_TIME` structure. The members of this structure are described in [Section 25.2.5, “C API Prepared Statement Data types”](#).

To send temporal data values, create a prepared statement using `mysql_stmt_prepare()`. Then, before calling `mysql_stmt_execute()` to execute the statement, use the following procedure to set up each temporal parameter:

1. In the `MYSQL_BIND` structure associated with the data value, set the `buffer_type` member to the type that indicates what kind of temporal value you're sending. For `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` values, set `buffer_type` to `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, respectively.
2. Set the `buffer` member of the `MYSQL_BIND` structure to the address of the `MYSQL_TIME` structure in which you pass the temporal value.
3. Fill in the members of the `MYSQL_TIME` structure that are appropriate for the type of temporal value to be passed.

Use `mysql_stmt_bind_param()` to bind the parameter data to the statement. Then you can call `mysql_stmt_execute()`.

To retrieve temporal values, the procedure is similar, except that you set the `buffer_type` member to the type of value you expect to receive, and the `buffer` member to the address of a `MYSQL_TIME` structure into which the returned value should be placed. Use `mysql_bind_results()` to bind the buffers to the statement after calling `mysql_stmt_execute()` and before fetching the results.

Here is a simple example that inserts `DATE`, `TIME`, and `TIMESTAMP` data. The `mysql` variable is assumed to be a valid connection handle.

```

MYSQL_TIME  ts;
MYSQL_BIND  bind[3];
MYSQL_STMT  *stmt;

strmov(query, "INSERT INTO test_table(date_field, time_field, \
            timestamp_field) VALUES(?,?,?)");

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
}

```

```

    exit(0);
}
if (mysql_stmt_prepare(mysql, query, strlen(query)))
{
    fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* set up input buffers for all 3 parameters */
bind[0].buffer_type= MYSQL_TYPE_DATE;
bind[0].buffer= (char *)&ts;
bind[0].is_null= 0;
bind[0].length= 0;
...
bind[1]= bind[2]= bind[0];
...

mysql_stmt_bind_param(stmt, bind);

/* supply the data to be sent in the ts structure */
ts.year= 2002;
ts.month= 02;
ts.day= 03;

ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_stmt_execute(stmt);
..

```

25.2.11. C API Threaded Function Descriptions

You need to use the following functions when you want to create a threaded client. See [Section 25.2.16, “How to Make a Threaded Client”](#).

25.2.11.1. `my_init()`

```
void my_init(void)
```

Description

`my_init()` initializes some global variables that MySQL needs. If you are using a thread-safe client library, it also calls `mysql_thread_init()` for this thread.

It is necessary for `my_init()` to be called early in the initialization phase of a program's use of the MySQL library. However, `my_init()` is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()`, and `mysql_connect()`. If you ensure that your program invokes one of those functions before any other MySQL calls, there is no need to invoke `my_init()` explicitly.

To access `my_init()`, your program must include the `my_sys.h` header file:

```
#include <my_sys.h>
```

Return Values

None.

25.2.11.2. `mysql_thread_init()`

```
my_bool mysql_thread_init(void)
```

Description

This function must be called early within each created thread to initialize thread-specific variables. However, you may not necessarily need to invoke it explicitly: `mysql_thread_init()` is automatically called by `my_init()`, which itself is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()`, and `mysql_connect()`. If you invoke any of those functions, `mysql_thread_init()` will be called for you.

Return Values

Zero if successful. Non-zero if an error occurred.

25.2.11.3. `mysql_thread_end()`

```
void mysql_thread_end(void)
```

Description

This function needs to be called before calling `pthread_exit()` to free memory allocated by `mysql_thread_init()`.

Note that this `mysql_thread_end()` is *not invoked automatically by the client library*. It must be called explicitly to avoid a memory leak.

Return Values

None.

25.2.11.4. `mysql_thread_safe()`

```
unsigned int mysql_thread_safe(void)
```

Description

This function indicates whether the client library is compiled as thread-safe.

Return Values

1 if the client library is thread-safe, 0 otherwise.

25.2.12. C API Embedded Server Function Descriptions

MySQL applications can be written to use an embedded server. See [Section 25.1, “libmysqld, the Embedded MySQL Server Library”](#). To write such an application, you must link it against the `libmysqld` library by using the `-lmysqld` flag rather than linking it against the `libmysqlclient` client library by using the `-libmysqlclient` flag. However, the calls to initialize and finalize the library are the same whether you write a client application or one that uses the embedded server: Call `mysql_library_init()` to initialize the library and `mysql_library_end()` when you are done with it. See [Section 25.2.2, “C API Function Overview”](#).

25.2.12.1. `mysql_server_init()`

```
int mysql_server_init(int argc, char **argv, char **groups)
```

Description

This function initializes the MySQL library, which must be done before you call any other MySQL function. However, `mysql_server_init()` is deprecated and you should call `mysql_library_init()` instead. See [Section 25.2.3.40, “mysql_library_init\(\)”](#).

Return Values

Zero if successful. Non-zero if an error occurred.

25.2.12.2. `mysql_server_end()`

```
void mysql_server_end(void)
```

Description

This function finalizes the MySQL library, which should be done when you are done using the library. However, `mysql_server_end()` is deprecated and `mysql_library_end()` should be used instead. See [Section 25.2.3.39, “mysql_library_end\(\)”](#).

Return Values

None.

25.2.13. Controlling Automatic Reconnect Behavior

The MySQL client library can perform an automatic reconnect to the server if it finds that the connection is down when you attempt to send a statement to the server to be executed. In this case, the library tries once to reconnect to the server and send the statement again.

If it is important for your application to know that the connection has been dropped (so that it can exit or take action to adjust for the loss of state information), be sure to disable auto-reconnect. This can be done explicitly by calling `mysql_options()` with the `MYSQL_OPT_RECONNECT` option:

```
my_bool reconnect = 0;
mysql_options(&mysql, MYSQL_OPT_RECONNECT, &reconnect);
```

In MySQL 5.1, auto-reconnect is disabled by default.

Some client programs might provide the capability of controlling automatic reconnection. For example, `mysql` reconnects by default, but the `--skip-reconnect` option can be used to suppress this behavior.

Automatic reconnection can be convenient because you need not implement your own reconnect code, but if a reconnection does occur, several aspects of the connection state are reset and your application will not know about it. The connection-related state is affected as follows:

- Any active transactions are rolled back and autocommit mode is reset.
- All table locks are released.
- All `TEMPORARY` tables are closed (and dropped).
- Session variables are reinitialized to the values of the corresponding variables. This also affects variables that are set implicitly by statements such as `SET NAMES`.
- User variable settings are lost.
- Prepared statements are released.
- `HANDLER` variables are closed.
- The value of `LAST_INSERT_ID()` is reset to 0.
- Locks acquired with `GET_LOCK()` are released.
- `mysql_ping()` does not attempt a reconnection if the connection is down. It returns an error instead.

25.2.14. Common Questions and Problems When Using the C API

MySQL Enterprise

Subscribers to MySQL Enterprise will find articles about the C API in the MySQL Knowledge Base. Access to the Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

25.2.14.1. Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success

It is possible for `mysql_store_result()` to return `NULL` following a successful call to `mysql_query()`. When this happens, it means one of the following conditions occurred:

- There was a `malloc()` failure (for example, if the result set was too large).
- The data couldn't be read (an error occurred on the connection).
- The query returned no data (for example, it was an `INSERT`, `UPDATE`, or `DELETE`).

You can always check whether the statement should have produced a non-empty result by calling `mysql_field_count()`. If `mysql_field_count()` returns zero, the result is empty and the last query was a statement that does not return values (for example, an `INSERT` or a `DELETE`). If `mysql_field_count()` returns a non-zero value, the statement should have produced a non-empty result. See the description of the `mysql_field_count()` function for an example.

You can test for an error by calling `mysql_error()` or `mysql_errno()`.

25.2.14.2. What Results You Can Get from a Query

In addition to the result set returned by a query, you can also get the following information:

- `mysql_affected_rows()` returns the number of rows affected by the last query when doing an `INSERT`, `UPDATE`, or `DELETE`.

For a fast re-create, use `TRUNCATE TABLE`.

- `mysql_num_rows()` returns the number of rows in a result set. With `mysql_store_result()`, `mysql_num_rows()` may be called as soon as `mysql_store_result()` returns. With `mysql_use_result()`, `mysql_num_rows()` may be called only after you have fetched all the rows with `mysql_fetch_row()`.
- `mysql_insert_id()` returns the ID generated by the last query that inserted a row into a table with an `AUTO_INCREMENT` index. See Section 25.2.3.37, “`mysql_insert_id()`”.
- Some queries (`LOAD DATA INFILE ...`, `INSERT INTO ... SELECT ...`, `UPDATE`) return additional information. The result is returned by `mysql_info()`. See the description for `mysql_info()` for the format of the string that it returns. `mysql_info()` returns a `NULL` pointer if there is no additional information.

25.2.14.3. How to Get the Unique ID for the Last Inserted Row

If you insert a record into a table that contains an `AUTO_INCREMENT` column, you can obtain the value stored into that column by calling the `mysql_insert_id()` function.

You can check from your C applications whether a value was stored in an `AUTO_INCREMENT` column by executing the following code (which assumes that you've checked that the statement succeeded). It determines whether the query was an `INSERT` with an `AUTO_INCREMENT` index:

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

When a new `AUTO_INCREMENT` value has been generated, you can also obtain it by executing a `SELECT LAST_INSERT_ID()` statement with `mysql_query()` and retrieving the value from the result set returned by the statement.

When inserting multiple values, the last automatically incremented value is returned.

For `LAST_INSERT_ID()`, the most recently generated ID is maintained in the server on a per-connection basis. It is not changed by another client. It is not even changed if you update another `AUTO_INCREMENT` column with a non-magic value (that is, a value that is not `NULL` and not `0`). Using `LAST_INSERT_ID()` and `AUTO_INCREMENT` columns simultaneously from multiple clients is perfectly valid. Each client will receive the last inserted ID for the last statement *that* client executed.

If you want to use the ID that was generated for one table and insert it into a second table, you can use SQL statements like this:

```
INSERT INTO foo (auto,text)
VALUES(NULL,'text');           # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
VALUES(LAST_INSERT_ID(),'text'); # use ID in second table
```

Note that `mysql_insert_id()` returns the value stored into an `AUTO_INCREMENT` column, whether that value is automatically generated by storing `NULL` or `0` or was specified as an explicit value. `LAST_INSERT_ID()` returns only automatically generated `AUTO_INCREMENT` values. If you store an explicit value other than `NULL` or `0`, it does not affect the value returned by `LAST_INSERT_ID()`.

For more information on obtaining the last ID in an `AUTO_INCREMENT` column:

- For information on `LAST_INSERT_ID()`, which can be used within an SQL statement, see ??? [694].
- For information on `mysql_insert_id()`, the function you use from within the C API, see Section 25.2.3.37, “`mysql_insert_id()`”.
- For information on obtaining the auto incremented value when using Connector/J see Section 26.4.5, “Connector/J Notes and Tips”.
- For information on obtaining the auto incremented value when using Connector/ODBC see Section 26.1.6.1.1, “Obtaining Auto-Increment Values”.

25.2.14.4. Problems Linking with the C API

When linking with the C API, the following errors may occur on some systems:

```
gcc -g -o client test.o -L/usr/local/lib/mysql \
    -lmysqlclient -lsocket -lnsl

Undefined      first referenced
symbol         in file
floor          /usr/local/lib/mysql/libmysqlclient.a(password.o)
ld: fatal: Symbol referencing errors. No output written to client
```

If this happens on your system, you must include the math library by adding `-lm` to the end of the compile/link line.

25.2.15. Building Client Programs

If you compile MySQL clients that you've written yourself or that you obtain from a third-party, they must be linked using the `-lmysqlclient -lz` options in the link command. You may also need to specify a `-L` option to tell the linker where to find the library. For example, if the library is installed in `/usr/local/mysql/lib`, use `-L/usr/local/mysql/lib -lmysqlclient -lz` in the link command.

For clients that use MySQL header files, you may need to specify an `-I` option when you compile them (for example, `-I/usr/local/mysql/include`), so that the compiler can find the header files.

To make it simpler to compile MySQL programs on Unix, we have provided the `mysql_config` script for you. See Section 25.9.2, “`mysql_config` — Get Compile Options for Compiling Clients”.

You can use it to compile a MySQL client as follows:

```
CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname `"$CFG" --cflags` progname.c `"$CFG" --libs`"
```

The `sh -c` is needed to get the shell not to treat the output from `mysql_config` as one word.

MySQL Enterprise

Subscribers to MySQL Enterprise will find an example client program in the Knowledge Base article, [Sample C program using the embedded MySQL server library](http://www.mysql.com/products/enterprise/advisors.html). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

25.2.16. How to Make a Threaded Client

The client library is almost thread-safe. The biggest problem is that the subroutines in `net.c` that read from sockets are not interrupt safe. This was done with the thought that you might want to have your own alarm that can break a long read to a server. If you install interrupt handlers for the `SIGPIPE` interrupt, the socket handling should be thread-safe.

To avoid aborting the program when a connection terminates, MySQL blocks `SIGPIPE` on the first call to `mysql_library_init()`, `mysql_init()`, or `mysql_connect()`. If you want to use your own `SIGPIPE` handler, you should first call `mysql_library_init()` and then install your handler.

In the older binaries that we distribute on our Web site (<http://www.mysql.com/>), client libraries other than those for Windows are not

normally compiled with the thread-safe option. Newer binary distributions should have both a normal and a thread-safe client library.

To get a threaded client where you can interrupt the client from other threads and set timeouts when talking with the MySQL server, you should use the `net_serv.o` code that the server uses and the `-lmysys`, `-lmystrings`, and `-ldbug` libraries.

If you don't need interrupts or timeouts, you can just compile a thread-safe client library (`mysqlclient_r`) and use this. See [Section 25.2, “MySQL C API”](#). In this case, you don't have to worry about the `net_serv.o` object file or the other MySQL libraries.

When using a threaded client and you want to use timeouts and interrupts, you can make great use of the routines in the `thr_alarm.c` file. If you are using routines from the `mysys` library, the only thing you must remember is to call `my_init()` first! See [Section 25.2.11, “C API Threaded Function Descriptions”](#).

In all cases, be sure to initialize the client library by calling `mysql_library_init()` before calling any other MySQL functions. When you are done with the library, call `mysql_library_end()`.

`mysql_real_connect()` is not thread-safe by default. The following notes describe how to compile a thread-safe client library and use it in a thread-safe manner. (The notes below for `mysql_real_connect()` also apply to `mysql_connect()` as well, although `mysql_connect()` is deprecated.)

To make `mysql_real_connect()` thread-safe, you must configure your MySQL distribution with this command:

```
shell> ./configure --enable-thread-safe-client
```

Then recompile the distribution to create a thread-safe client library, `libmysqlclient_r`. (Assuming that your operating system has a thread-safe `gethostbyname_r()` function.) This library is thread-safe per connection. You can let two threads share the same connection with the following caveats:

- Two threads can't send a query to the MySQL server at the same time on the same connection. In particular, you have to ensure that between calls to `mysql_query()` and `mysql_store_result()` no other thread is using the same connection.
- Many threads can access different result sets that are retrieved with `mysql_store_result()`.
- If you use `mysql_use_result`, you must ensure that no other thread is using the same connection until the result set is closed. However, it really is best for threaded clients that share the same connection to use `mysql_store_result()`.
- If you want to use multiple threads on the same connection, you must have a mutex lock around your pair of `mysql_query()` and `mysql_store_result()` calls. Once `mysql_store_result()` is ready, the lock can be released and other threads may query the same connection.
- If you use POSIX threads, you can use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to establish and release a mutex lock.

You need to know the following if you have a thread that is calling MySQL functions which did not create the connection to the MySQL database:

When you call `mysql_init()` or `mysql_connect()`, MySQL creates a thread-specific variable for the thread that is used by the debug library (among other things).

If you call a MySQL function, before the thread has called `mysql_init()` or `mysql_connect()`, the thread does not have the necessary thread-specific variables in place and you are likely to end up with a core dump sooner or later.

To get things to work smoothly you have to do the following:

1. Call `mysql_library_init()` before any other MySQL functions. It is not thread-safe, so call it before threads are created, or protect the call with a mutex.
2. Arrange for `mysql_thread_init()` to be called early in the thread handler before calling any MySQL function. If you call `mysql_init()` or `mysql_connect()`, they will call `mysql_thread_init()` for you.
3. In the thread, call `mysql_thread_end()` before calling `pthread_exit()`. This frees the memory used by MySQL thread-specific variables.

If “undefined symbol” errors occur when linking your client with `libmysqlclient_r`, in most cases this is because you haven't in-

cluded the thread libraries on the link/compile command.

25.3. MySQL PHP API

PHP is a server-side, HTML-embedded scripting language that may be used to create dynamic Web pages. It is available for most operating systems and Web servers, and can access most common databases, including MySQL. PHP may be run as a separate program or compiled as a module for use with the Apache Web server.

PHP actually provides two different MySQL API extensions:

- **mysql**: Available for PHP versions 4 and 5, this extension is intended for use with MySQL versions prior to MySQL 4.1. This extension does not support the improved authentication protocol used in MySQL 5.1, nor does it support prepared statements or multiple statements. If you wish to use this extension with MySQL 5.1, you will likely want to configure the MySQL server to use the `--old-passwords` option (see [Section B.1.2.3, “Client does not support authentication protocol”](#)). This extension is documented on the PHP Web site at <http://php.net/mysql>.
- **mysqli** - Stands for “MySQL, Improved”; this extension is available only in PHP 5. It is intended for use with MySQL 4.1.1 and later. This extension fully supports the authentication protocol used in MySQL 5.1, as well as the Prepared Statements and Multiple Statements APIs. In addition, this extension provides an advanced, object-oriented programming interface. You can read the documentation for the **mysqli** extension at <http://php.net/mysqli>. A helpful article can be found at <http://www.zend.com/php5/articles/php5-mysqli.php>.

If you're experiencing problems with enabling both the **mysql** and the **mysqli** extension when building PHP on Linux yourself, see [Section 25.3.2, “Enabling Both mysql and mysqli in PHP”](#).

The PHP distribution and documentation are available from the [PHP Web site](#). MySQL provides the **mysql** and **mysqli** extensions for the Windows operating system on <http://dev.mysql.com/downloads/connector/php/>. You can find information why you should preferably use the extensions provided by MySQL on that page.

MySQL Enterprise

MySQL Enterprise subscribers will find more information about MySQL and PHP in the Knowledge Base articles found at [PHP](#). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

25.3.1. Common Problems with MySQL and PHP

- **Error: Maximum Execution Time Exceeded**: This is a PHP limit; go into the `php.ini` file and set the maximum execution time up from 30 seconds to something higher, as needed. It is also not a bad idea to double the RAM allowed per script to 16MB instead of 8MB.
- **Fatal error: Call to unsupported or undefined function mysql_connect() in ...**: This means that your PHP version isn't compiled with MySQL support. You can either compile a dynamic MySQL module and load it into PHP or recompile PHP with built-in MySQL support. This process is described in detail in the PHP manual.
- **Error: Undefined reference to 'uncompress'**: This means that the client library is compiled with support for a compressed client/server protocol. The fix is to add `-lz` last when linking with `-lmysqlclient`.
- **Error: Client does not support authentication protocol**: This is most often encountered when trying to use the older **mysql** extension with MySQL 4.1.1 and later. Possible solutions are: downgrade to MySQL 4.0; switch to PHP 5 and the newer **mysqli** extension; or configure the MySQL server with `--old-passwords`. (See [Section B.1.2.3, “Client does not support authentication protocol”](#), for more information.)

Those with PHP4 legacy code can make use of a compatibility layer for the old and new MySQL libraries, such as this one: <http://www.coggeshall.org/oss/mysql2i>.

25.3.2. Enabling Both **mysql** and **mysqli** in PHP

If you're experiencing problems with enabling both the **mysql** and the **mysqli** extension when building PHP on Linux yourself, you should try the following procedure.

1. Configure PHP like this:

```
./configure --with-mysql=/usr/bin/mysql_config --with-mysql=/usr
```

2. Edit the `Makefile` and search for a line that starts with `EXTRA_LIBS`. It might look like this (all on one line):

```
EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl -lnsl  
-lxml2 -lz -lm -lxml2 -lz -lm -lmysqlclient -lz -lcrypt -lnsl -lm  
-lxml2 -lz -lm -lcrypt -lxml2 -lz -lm -lcrypt
```

Remove all duplicates, so that the line looks like this (all on one line):

```
EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl -lnsl  
-lxml2
```

3. Build and install PHP:

```
make  
make install
```

MySQL Enterprise

MySQL Enterprise subscribers will find more information about the `mysql` extension in the Knowledge Base articles found at [mysql](http://mysql.com). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

25.4. MySQL Perl API

The Perl `DBI` module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI, you must install the `DBI` module, as well as a DataBase Driver (DBD) module for each type of server you want to access. For MySQL, this driver is the `DBD::mysql` module.

Perl DBI is the recommended Perl interface. It replaces an older interface called `mysqlperl`, which should be considered obsolete.

Installation instructions for Perl DBI support are given in [Section 2.15, “Perl Installation Notes”](#).

DBI information is available at the command line, online, or in printed form:

- Once you have the `DBI` and `DBD::mysql` modules installed, you can get information about them at the command line with the `perldoc` command:

```
shell> perldoc DBI  
shell> perldoc DBI::FAQ  
shell> perldoc DBD::mysql
```

You can also use `pod2man`, `pod2html`, and so forth to translate this information into other formats.

- For online information about Perl DBI, visit the DBI Web site, <http://dbi.perl.org/>. That site hosts a general DBI mailing list. MySQL AB hosts a list specifically about `DBD::mysql`; see [Section 1.7.1, “MySQL Mailing Lists”](#).
- For printed information, the official DBI book is *Programming the Perl DBI* (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000). Information about the book is available at the DBI Web site, <http://dbi.perl.org/>.

For information that focuses specifically on using DBI with MySQL, see *MySQL and Perl for the Web* (Paul DuBois, New Riders, 2001). This book's Web site is <http://www.kitebird.com/mysql-perl/>.

25.5. MySQL C++ API

`MySQL++` is a MySQL API for C++. Warren Young has taken over this project. More information can be found at <http://tangentsoft.net/mysql++/doc>.

MySQL Enterprise

MySQL Enterprise subscribers will find more information about using C++ with the MySQL API in the MySQL Knowledge Base. articles found at [C++](#). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

25.6. MySQL Python API

`MySQLdb` provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at <http://sourceforge.net/projects/mysql-python/>.

MySQL Enterprise

MySQL Enterprise subscribers will find more information about using Python with the MySQL API in the MySQL Knowledge Base articles found at [Python](#). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

25.7. MySQL Tcl API

`MySQLtcl` is a simple API for accessing a MySQL database server from the Tcl programming language. It can be found at <http://www.xdobry.de/mysqltcl/>.

25.8. MySQL Eiffel Wrapper

Eiffel MySQL is an interface to the MySQL database server using the Eiffel programming language, written by Michael Ravits. It can be found at <http://efsa.sourceforge.net/archive/ravits/mysql.htm>.

25.9. MySQL Program Development Utilities

This section describes some utilities that you may find useful when developing MySQL programs.

- `msql2mysql`

A shell script that converts `mSQL` programs to MySQL. It doesn't handle every case, but it gives a good start when converting.

- `mysql_config`

A shell script that produces the option values needed when compiling MySQL programs.

25.9.1. `msql2mysql` — Convert mSQL Programs for Use with MySQL

Initially, the MySQL C API was developed to be very similar to that for the mSQL database system. Because of this, mSQL programs often can be converted relatively easily for use with MySQL by changing the names of the C API functions.

The `msql2mysql` utility performs the conversion of mSQL C API function calls to their MySQL equivalents. `msql2mysql` converts the input file in place, so make a copy of the original before converting it. For example, use `msql2mysql` like this:

```
shell> cp client-prog.c client-prog.c.orig
shell> msql2mysql client-prog.c
client-prog.c converted
```

Then examine `client-prog.c` and make any post-conversion revisions that may be necessary.

`msql2mysql` uses the `replace` utility to make the function name substitutions. See [Section 7.26, “replace — A String-Replacement Utility”](#).

25.9.2. `mysql_config` — Get Compile Options for Compiling Clients

`mysql_config` provides you with useful information for compiling your MySQL client and connecting it to MySQL.

`mysql_config` supports the following options:

- `--cflags`

Compiler flags to find include files and critical compiler flags and defines used when compiling the `libmysqlclient` library.

- `--include`

Compiler options to find MySQL include files. (Note that normally you would use `--cflags` instead of this option.)

- `--libmysqld-libs, --embedded`

Libraries and options required to link with the MySQL embedded server.

- `--libs`

Libraries and options required to link with the MySQL client library.

- `--libs_r`

Libraries and options required to link with the thread-safe MySQL client library.

- `--port`

The default TCP/IP port number, defined when configuring MySQL.

- `--socket`

The default Unix socket file, defined when configuring MySQL.

- `--version`

Version number for the MySQL distribution.

If you invoke `mysql_config` with no options, it displays a list of all options that it supports, and their values:

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
--cflags      [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
--include     [-I/usr/local/mysql/include/mysql]
--libs        [-L/usr/local/mysql/lib/mysql -lmysqlclient -lz
              -lcrypt -lnsl -lm -L/usr/lib -lssl -lcrypto]
--libs_r      [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
              -lpthread -lz -lcrypt -lnsl -lm -lpthread]
--socket      [/tmp/mysql.sock]
--port        [3306]
--version     [4.0.16]
--libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld -lpthread -lz
                 -lcrypt -lnsl -lm -lpthread -lrt]
```

You can use `mysql_config` within a command line to include the value that it displays for a particular option. For example, to compile a MySQL client program, use `mysql_config` as follows:

```
shell> CFG=/usr/local/mysql/bin/mysql_config
shell> sh -c "gcc -o progname ` $CFG --cflags ` progname.c ` $CFG --libs `"
```

When you use `mysql_config` this way, be sure to invoke it within backtick (``) characters. That tells the shell to execute it and substitute its output into the surrounding command.

Chapter 26. Connectors

This chapter describes MySQL Connectors, drivers that provide connectivity to the MySQL server for client programs. There are currently five MySQL Connectors:

- Connector/ODBC provides driver support for connecting to a MySQL server using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix and Mac OS X platforms.
- Connector/NET enables developers to create .NET applications that use data stored in a MySQL database. Connector/NET implements a fully-functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that want to use Connector/NET can be written in any of the supported .NET languages.
- The MySQL Visual Studio Plugin works with Connector/NET and Visual Studio 2005. The plugin is a MySQL DDEX Provider, which means that you can use the schema and data manipulation tools within Visual Studio to create and edit objects within a MySQL database.
- Connector/J provides driver support for connecting to MySQL from a Java application using the standard Java Database Connectivity (JDBC) API.
- Connector/MXJ is a tool that enables easy deployment and management of MySQL server and database through your Java application.
- Connector/PHP is a Windows-only connector for PHP that provides the `mysql` and `mysqli` extensions for use with MySQL 5.0.18 and later.

For information on connecting to a MySQL server using other languages and interfaces than those detailed above, including Perl, Python and PHP for other platforms and environments, please refer to the [Chapter 25, APIs and Libraries](#) chapter.

26.1. MySQL Connector/ODBC

The MySQL Connector/ODBC is the name for the family of MySQL ODBC drivers (previously called MyODBC drivers) that provide access to a MySQL database using the industry standard Open Database Connectivity (ODBC) API. This reference covers Connector/ODBC 3.51, a version of the API that provides ODBC 3.5x compliant access to a MySQL database.

The manual for versions of Connector/ODBC older than 3.51 can be located in the corresponding binary or source distribution.

For more information on the ODBC API standard and how to use it, refer to <http://www.microsoft.com/data/>.

The application development part of this reference assumes a good working knowledge of C, general DBMS knowledge, and finally, but not least, familiarity with MySQL. For more information about MySQL functionality and its syntax, refer to <http://dev.mysql.com/doc/>.

Typically, you need to install Connector/ODBC only on Windows machines. For Unix and Mac OS X you can use the native MySQL network or named pipe to communicate with your MySQL database. You may need Connector/ODBC for Unix or Mac OS X if you have an application that requires an ODBC interface to communicate with the database. Applications that require ODBC to communicate with MySQL include ColdFusion, Microsoft Office, and Filemaker Pro.

If you want to install the Connector/ODBC connector on a Unix host, then you must also install an ODBC manager.

Key topics:

- For help installing Connector/ODBC see [Section 26.1.2, “Connector/ODBC Installation”](#).
- For more information on connecting to a MySQL database from a Windows host using Connector/ODBC see [Section 26.1.4.2, “Step-by-step Guide to Connecting to a MySQL Database through Connector/ODBC”](#).
- If you want to use Microsoft Access as an interface to a MySQL database using Connector/ODBC see [Section 26.1.4.4, “Using Connector/ODBC with Microsoft Access”](#).
- General tips on using Connector/ODBC, including obtaining the last auto increment ID see [Section 26.1.6.1, “Connector/ODBC General Functionality”](#).

- For tips and common questions on using Connector/ODBC with specific application see [Section 26.1.6.2, “Connector/ODBC Application Specific Tips”](#).
- For a general list of Frequently Asked Questions see [Section 26.1.6.3, “Connector/ODBC Errors and Resolutions \(FAQ\)”](#).
- Additional support when using Connector/ODBC is available, see [Section 26.1.7, “Connector/ODBC Support”](#).

MySQL Enterprise

MySQL Enterprise subscribers will find more information about MySQL and ODBC in the Knowledge Base articles about [ODBC](#). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

26.1.1. Introduction to Connector/ODBC

ODBC (Open Database Connectivity) provides a way for client programs to access a wide range of databases or data sources. ODBC is a standardized API that allows connections to SQL database servers. It was developed according to the specifications of the SQL Access Group and defines a set of function calls, error codes, and data types that can be used to develop database-independent applications. ODBC usually is used when database independence or simultaneous access to different data sources is required.

For more information about ODBC, refer to <http://www.microsoft.com/data/>.

26.1.1.1. Connector/ODBC Versions

There are currently two version of Connector/ODBC available:

- Connector/ODBC 5.0, currently in beta status, has been designed to extend the functionality of the Connector/ODBC 3.51 driver and incorporate full support for the functionality in the MySQL 5.0 server release, including stored procedures and views. Applications using Connector/ODBC 3.51 will be compatible with Connector/ODBC 5.0, while being able to take advantage of the new features. Features and functionality of the Connector/ODBC 5.0 driver are not currently included in this guide.
- Connector/ODBC 3.51 is the current release of the 32-bit ODBC driver, also known as the MySQL ODBC 3.51 driver. This version is enhanced compared to the older Connector/ODBC 2.50 driver. It has support for ODBC 3.5x specification level 1 (complete core API + level 2 features) in order to continue to provide all functionality of ODBC for accessing MySQL.
- MyODBC 2.50 is the previous version of the 32-bit ODBC driver from MySQL AB that is based on ODBC 2.50 specification level 0 (with level 1 and 2 features). Information about the MyODBC 2.50 driver is included in this guide for the purposes of comparison only.

Note

From this section onward, the primary focus of this guide is the Connector/ODBC 3.51 driver. More information about the MyODBC 2.50 driver in the documentation included in the installation packages for that version. If there is a specific issue (error or known problem) that only affects the 2.50 version, it may be included here for reference.

Note

Version numbers for MySQL products are formatted as X.X.X. However, Windows tools (Control Panel, properties display) may show the version numbers as XX.XX.XX. For example, the official MySQL formatted version number 5.0.9 may be displayed by Windows tools as 5.00.09. The two versions are the same; only the number display format is different.

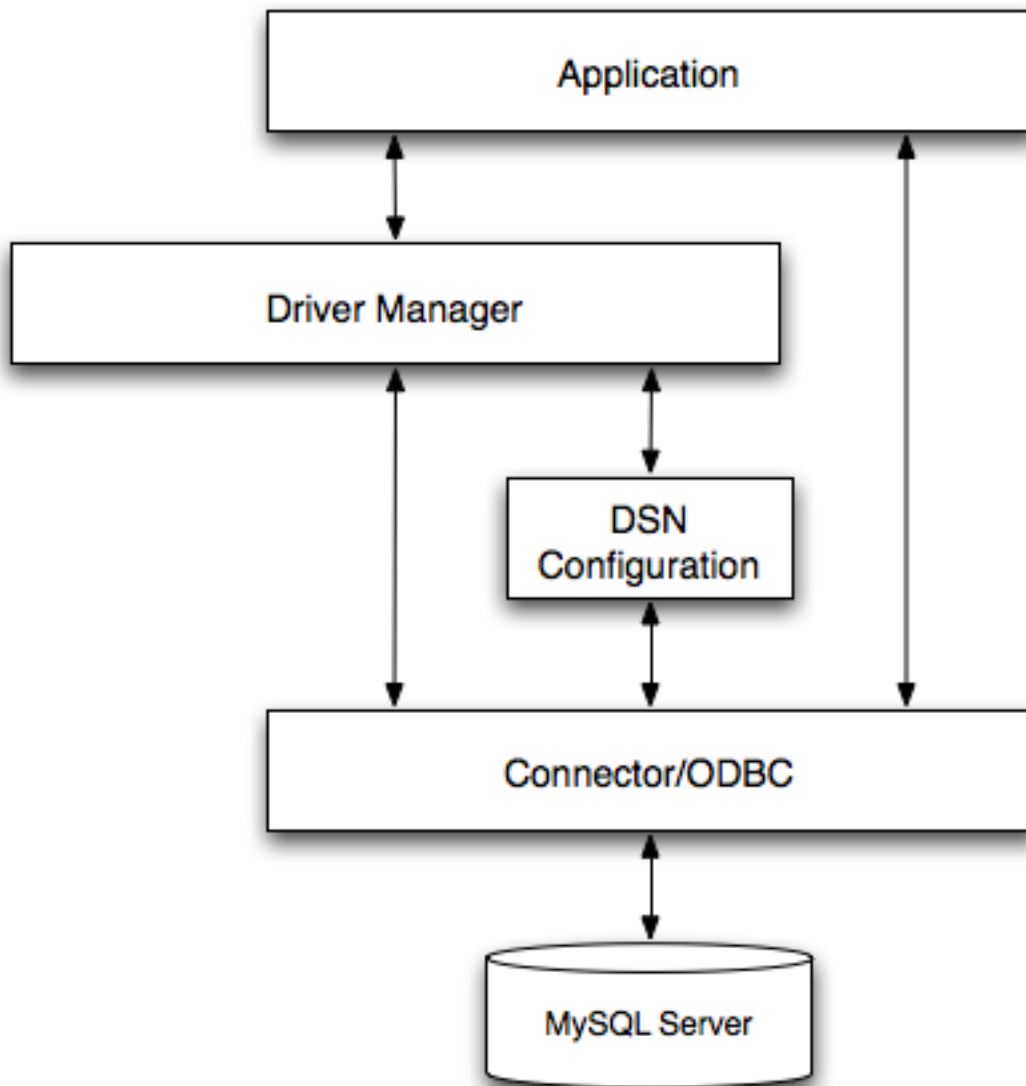
26.1.1.2. General Information About ODBC and Connector/ODBC

Open Database Connectivity (ODBC) is a widely accepted application-programming interface (API) for database access. It is based on the Call-Level Interface (CLI) specifications from X/Open and ISO/IEC for database APIs and uses Structured Query Language (SQL) as its database access language.

A survey of ODBC functions supported by Connector/ODBC is given at [Section 26.1.5.1, “Connector/ODBC API Reference”](#). For general information about ODBC, see <http://www.microsoft.com/data/>.

26.1.1.2.1. Connector/ODBC Architecture

The Connector/ODBC architecture is based on five components, as shown in the following diagram:



- **Application:**

The Application uses the ODBC API to access the data from the MySQL server. The ODBC API in turn uses the communicates with the Driver Manager. The Application communicates with the Driver Manager using the standard ODBC calls. The Application does not care where the data is stored, how it is stored, or even how the system is configured to access the data. It needs to know only the Data Source Name (DSN).

A number of tasks are common to all applications, no matter how they use ODBC. These tasks are:

- Selecting the MySQL server and connecting to it
- Submitting SQL statements for execution
- Retrieving results (if any)

- Processing errors
- Committing or rolling back the transaction enclosing the SQL statement
- Disconnecting from the MySQL server

Because most data access work is done with SQL, the primary tasks for applications that use ODBC are submitting SQL statements and retrieving any results generated by those statements.

- **Driver manager:**

The Driver Manager is a library that manages communication between application and driver or drivers. It performs the following tasks:

- Resolves Data Source Names (DSN). The DSN is a configuration string that identifies a given database driver, database, database host and optionally authentication information that enables an ODBC application to connect to a database using a standardized reference.

Because the database connectivity information is identified by the DSN, any ODBC compliant application can connect to the data source using the same DSN reference. This eliminates the need to separately configure each application that needs access to a given database; instead you instruct the application to use a pre-configured DSN.

- Loading and unloading of the driver required to access a specific database as defined within the DSN. For example, if you have configured a DSN that connects to a MySQL database then the driver manager will load the Connector/ODBC driver to enable the ODBC API to communicate with the MySQL host.
- Processes ODBC function calls or passes them to the driver for processing.

- **Connector/ODBC Driver:**

The Connector/ODBC driver is a library that implements the functions supported by the ODBC API. It processes ODBC function calls, submits SQL requests to MySQL server, and returns results back to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by MySQL.

- **DSN Configuration:**

The ODBC configuration file stores the driver and database information required to connect to the server. It is used by the Driver Manager to determine which driver to be loaded according to the definition in the DSN. The driver uses this to read connection parameters based on the DSN specified. For more information, [Section 26.1.3, “Connector/ODBC Configuration”](#).

- **MySQL Server:**

The MySQL database where the information is stored. The database is used as the source of the data (during queries) and the destination for data (during inserts and updates).

26.1.1.2.2. ODBC Driver Managers

An ODBC Driver Manager is a library that manages communication between the ODBC-aware application and any drivers. Its main functionality includes:

- Resolving Data Source Names (DSN).
- Driver loading and unloading.
- Processing ODBC function calls or passing them to the driver.

Both Windows and Mac OS X include ODBC driver managers with the operating system. Most ODBC Driver Manager implementations also include an administration application that makes the configuration of DSN and drivers easier. Examples and information on these managers, including Unix ODBC driver managers are listed below:

- Microsoft Windows ODBC Driver Manager ([odbc32.dll](#)), <http://www.microsoft.com/data/>.

- Mac OS X includes [ODBC Administrator](#), a GUI application that provides a simpler configuration mechanism for the Unix iODBC Driver Manager. You can configure DSN and driver information either through ODBC Administrator or through the iODBC configuration files. This also means that you can test ODBC Administrator configurations using the `iodbctest` command. <http://www.apple.com>.
- [unixODBC](#) Driver Manager for Unix ([libodbc.so](#)). See <http://www.unixodbc.org>, for more information. The [unixODBC](#) Driver Manager includes the Connector/ODBC driver 3.51 in the installation package, starting with version [unixODBC 2.1.2](#).
- [iODBC](#) ODBC Driver Manager for Unix ([libiodbc.so](#)), see <http://www.iodbc.org>, for more information.

26.1.2. Connector/ODBC Installation

You can install the Connector/ODBC drivers using two different methods, a binary installation and a source installation. The binary installation is the easiest and most straightforward method of installation. Using the source installation methods should only be necessary on platforms where a binary installation package is not available, or in situations where you want to customize or modify the installation process or Connector/ODBC drivers before installation.

Where to Get Connector/ODBC

MySQL AB distributes all its products under the General Public License (GPL). You can get a copy of the latest version of Connector/ODBC binaries and sources from the MySQL AB Web site <http://dev.mysql.com/downloads/>.

For more information about Connector/ODBC, visit <http://www.mysql.com/products/myodbc/>.

For more information about licensing, visit <http://www.mysql.com/company/legal/licensing/>.

Supported Platforms

Connector/ODBC can be used on all major platforms supported by MySQL. You can install it on:

- Windows 95, 98, Me, NT, 2000, XP, and 2003
- All Unix-like Operating Systems, including: AIX, Amiga, BSDI, DEC, FreeBSD, HP-UX 10/11, Linux, NetBSD, OpenBSD, OS/2, SGI Irix, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64 Unix
- Mac OS X and Mac OS X Server

Using a binary distribution offers the most straightforward method for installing Connector/ODBC. If you want more control over the driver, the installation location and or to customize elements of the driver you will need to build and install from the source.

If a binary distribution is not available for a particular platform build the driver from the original source code. You can contribute the binaries you create to MySQL by sending a mail message to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com), so that it becomes available for other users.

For further instructions:

Platform	Binary	Source
Windows	Installation Instructions	Build Instructions
Unix/Linux	Installation Instructions	Build Instructions
Mac OS X	Installation Instructions	

26.1.2.1. Installing Connector/ODBC from a Binary Distribution on Windows

Before installing the Connector/ODBC drivers on Windows you should ensure that your Microsoft Data Access Components (MDAC) are up to date. You can obtain the latest version from the [Microsoft Data Access and Storage](#) Web site.

There are three available distribution types to use when installing for Windows. The contents in each case are identical, it is only the installation method which is different.

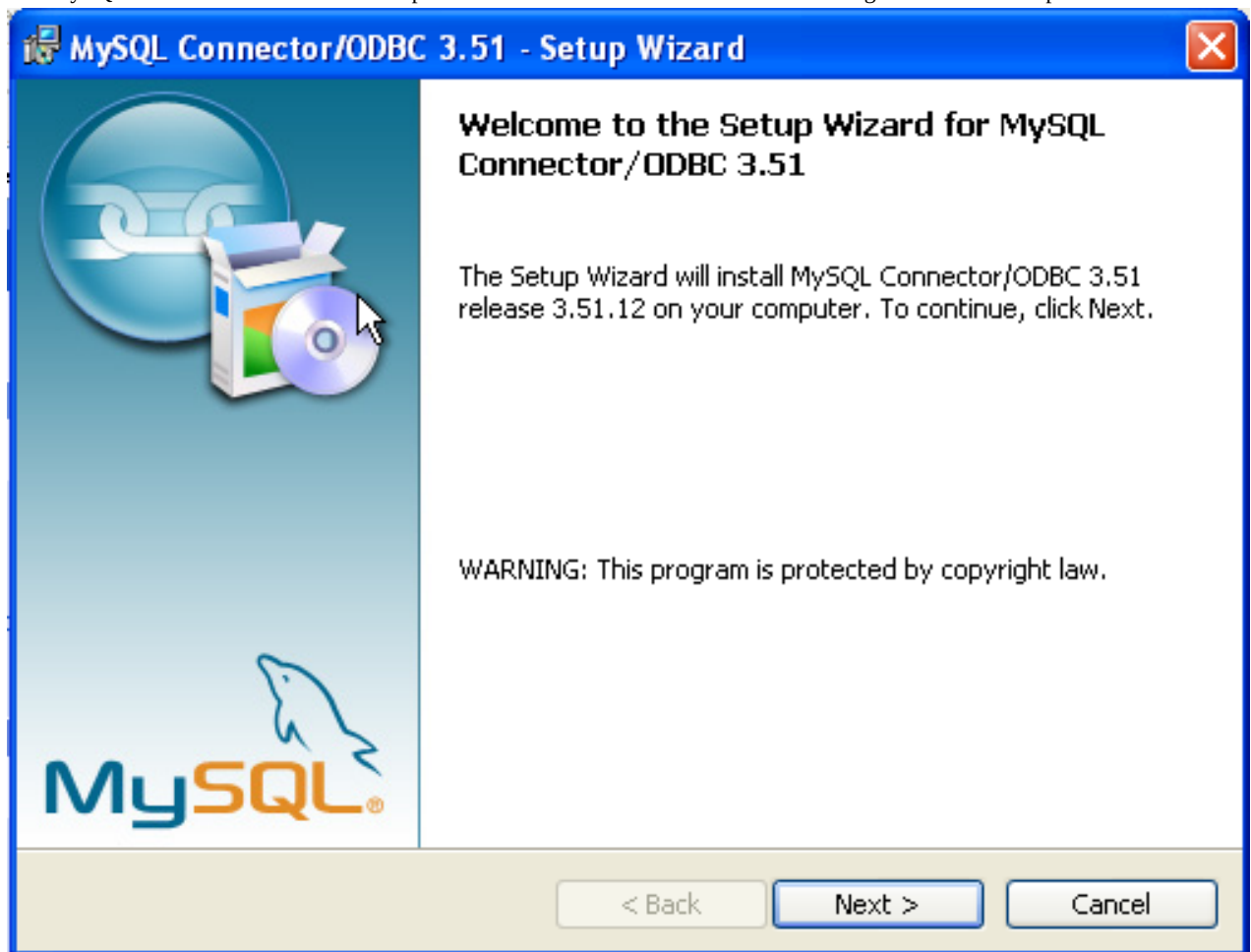
- Zipped installer consists of a Zipped package containing a standalone installation application. To install from this package, you must unzip the installer, and then run the installation application. See [Section 26.1.2.1.1, “Installing the Windows Connector/ODBC Driver using an installer”](#) to complete the installation.
- MSI installer, an installation file that can be used with the installer included in Windows 2000, Windows XP and Windows Server 2003. See [Section 26.1.2.1.1, “Installing the Windows Connector/ODBC Driver using an installer”](#) to complete the installation.
- Zipped DLL package, containing the DLL files that need must be manually installed. See [Section 26.1.2.1.2, “Installing the Windows Connector/ODBC Driver using the Zipped DLL package”](#) to complete the installation.

26.1.2.1.1. Installing the Windows Connector/ODBC Driver using an installer

The installer packages offer a very simple method for installing the Connector/ODBC drivers. If you have downloaded the zipped installer then you must extract the installer application. The basic installation process is identical for both installers.

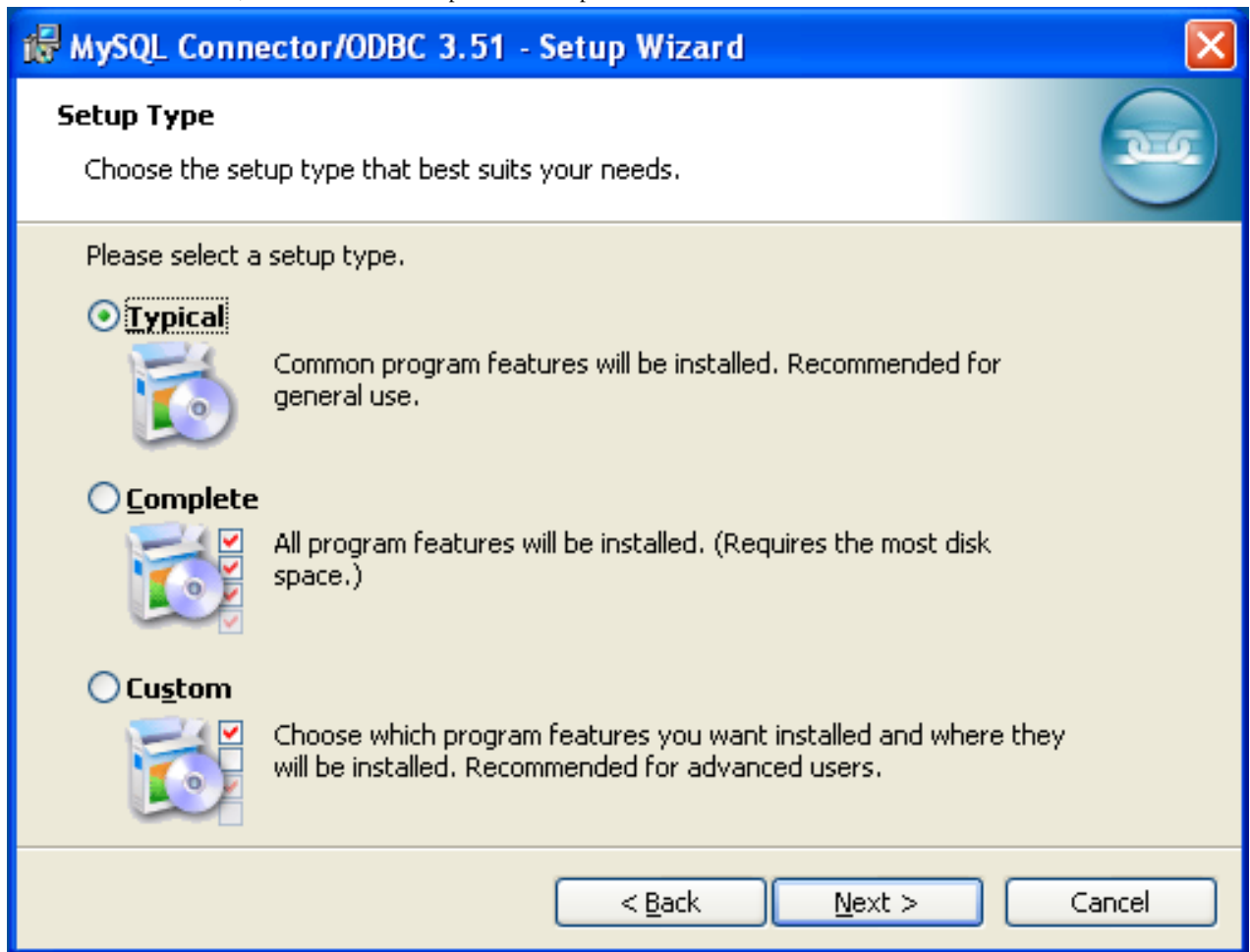
You should follow these steps to complete the installation:

1. Double click on the standalone installer that you extracted, or the MSI file you downloaded.
2. The MySQL Connector/ODBC 3.51 - Setup Wizard will start. Click the NEXT button to begin the installation process.

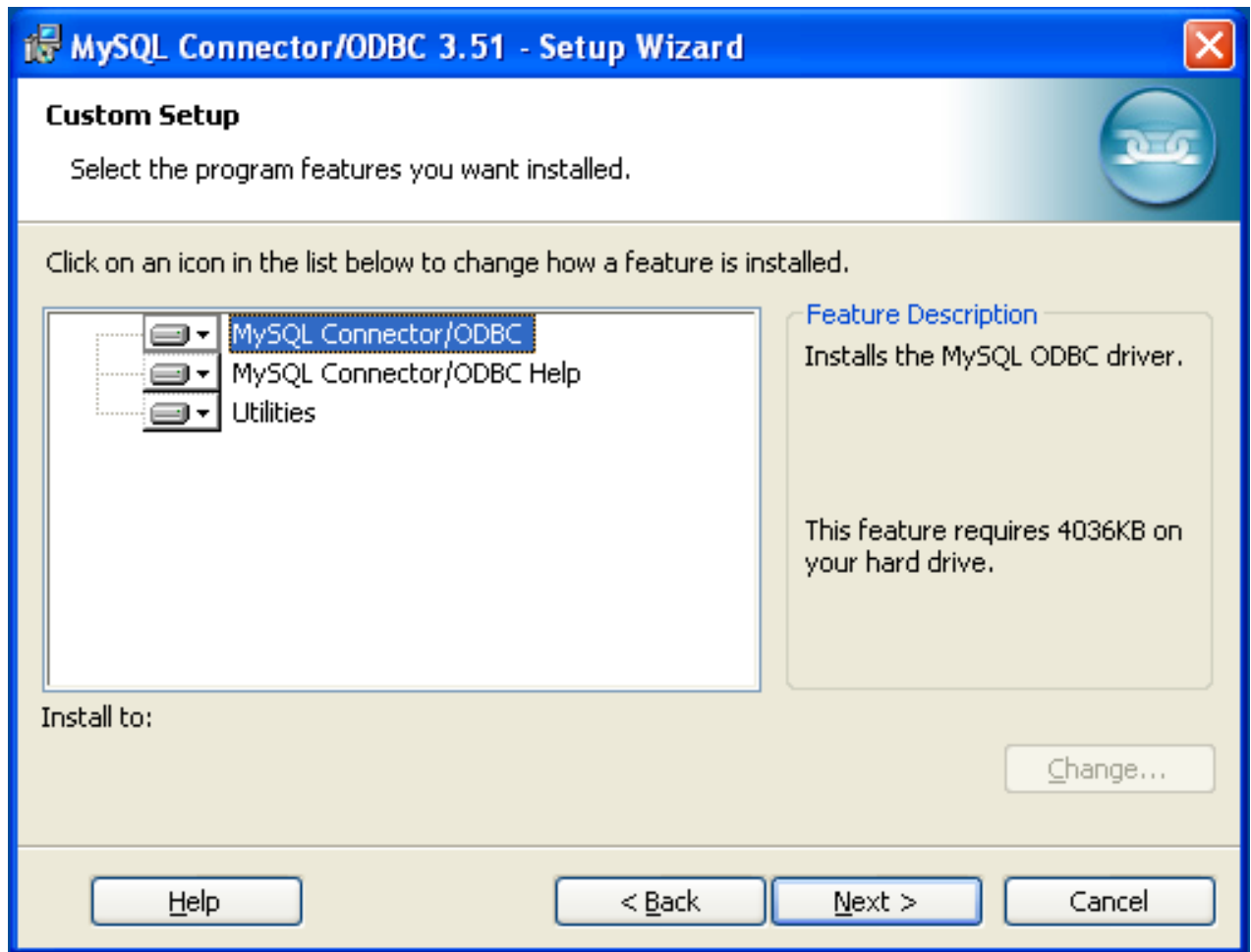


3. You will need to choose the installation type. The Typical installation provides the standard files you will need to connect to a MySQL database using ODBC. The Complete option installs all the available files, including debug and utility components. It is recommended you choose one of these two options to complete the installation. If choose one of these methods, click NEXT and then proceed to step 5.

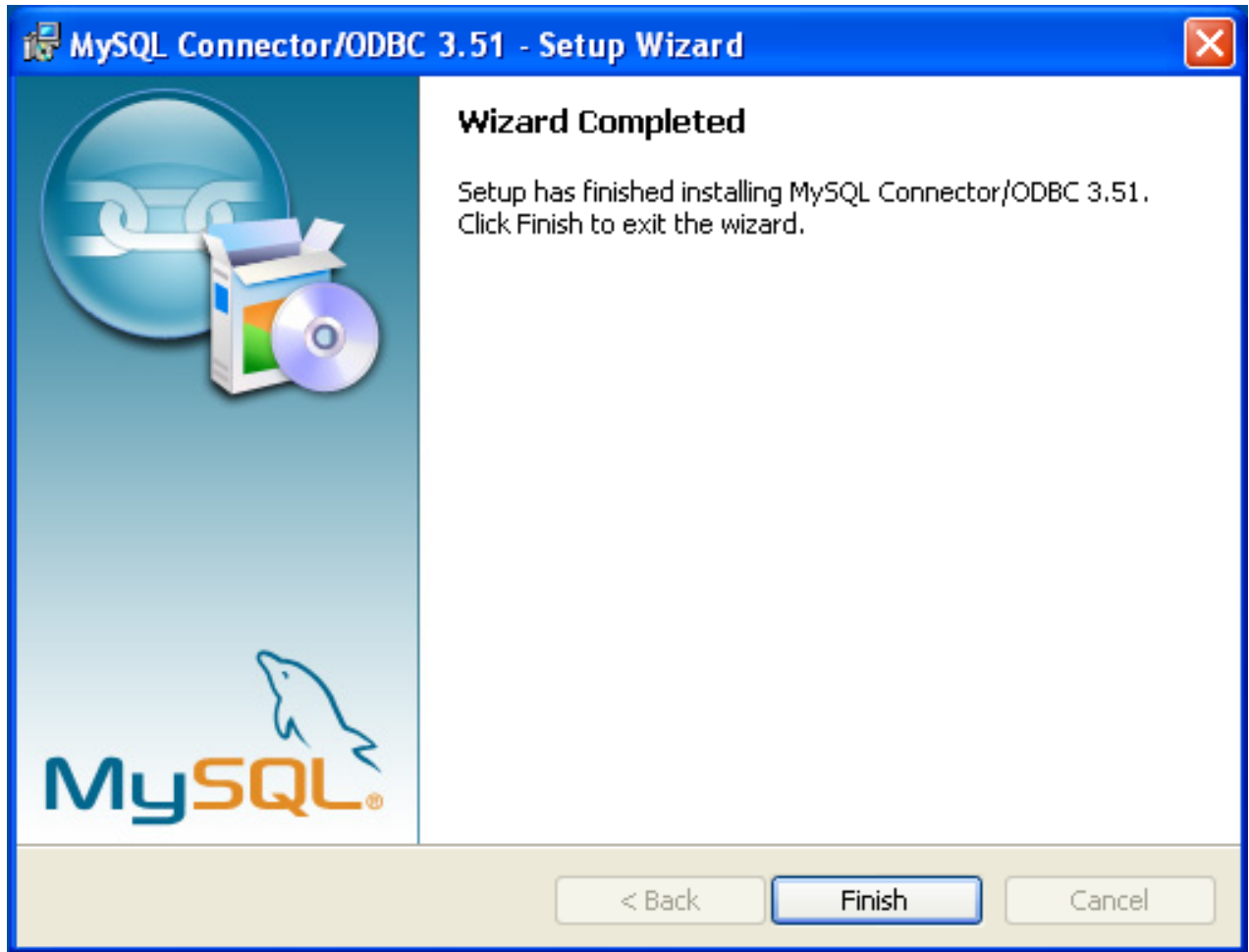
You may also choose a Custom installation, which enables you to select the individual components that you want to install. You have chosen this method, click NEXT and then proceed to step 4.



4. If you have chosen a custom installation, use the popups to select which components to install and then click NEXT to install the necessary files.



5. Once the files have copied to your machine, the installation is complete. Click FINISH to exit the installer.



Now the installation is complete, you can continue to configure your ODBC connections using [Section 26.1.3, “Connector/ODBC Configuration”](#).

26.1.2.1.2. Installing the Windows Connector/ODBC Driver using the Zipped DLL package

If you have downloaded the Zipped DLL package then you must install the individual files required for Connector/ODBC operation manually. Once you have unzipped the installation files, you can either perform this operation by hand, executing each statement individually, or you can use the included Batch file to perform an installation to the default locations.

To install using the Batch file:

1. Unzip the Connector/ODBC Zipped DLL package.
2. Open a Command Prompt.
3. Change to the directory created when you unzipped the Connector/ODBC Zipped DLL package.
4. Run `Install.bat`:

```
C:\> Install.bat
```

This will copy the necessary files into the default location, and then register the Connector/ODBC driver with the Windows ODBC manager.

If you want to copy the files to an alternative location - for example, to run or test different versions of the Connector/ODBC driver on the same machine, then you must copy the files by hand. It is however not recommended to install these files in a non-standard location. To copy the files by hand to the default installation location use the following steps:

1. Unzip the Connector/ODBC Zipped DLL package.
2. Open a Command Prompt.
3. Change to the directory created when you unzipped the Connector/ODBC Zipped DLL package.
4. Copy the library files to a suitable directory. The default is to copy them into the default Windows system directory `\Windows\System32`:

```
C:\> copy lib\myodbc3S.dll \Windows\System32
C:\> copy lib\myodbc3S.lib \Windows\System32
C:\> copy lib\myodbc3.dll \Windows\System32
C:\> copy lib\myodbc3.lib \Windows\System32
```

5. Copy the Connector/ODBC tools. These must be placed into a directory that is in the system `PATH`. The default is to install these into the Windows system directory `\Windows\System32`:

```
C:\> copy bin\myodbc3i.exe \Windows\System32
C:\> copy bin\myodbc3m.exe \Windows\System32
C:\> copy bin\myodbc3c.exe \Windows\System32
```

6. Optionally copy the help files. For these files to be accessible through the help system, they must be installed in the Windows system directory:

```
C:\> copy doc\*.hlp \Windows\System32
```

7. Finally, you must register the Connector/ODBC driver with the ODBC manager:

```
C:\> myodbc3i -a -d -t"MySQL ODBC 3.51 Driver;\
DRIVER=myodbc3.dll;SETUP=myodbc3S.dll"
```

You must change the references to the DLL files and command location in the above statement if you have not installed these files into the default location.

26.1.2.1.3. Handling Installation Errors

On Windows, you may get the following error when trying to install the older MyODBC 2.50 driver:

```
An error occurred while copying C:\WINDOWS\SYSTEM\MFC30.DLL.
Restart Windows and try installing again (before running any
applications which use ODBC)
```

The reason for the error is that another application is currently using the ODBC system. Windows may not allow you to complete the installation. In most cases, you can continue by pressing **Ignore** to copy the rest of the Connector/ODBC files and the final installation should still work. If it doesn't, the solution is to re-boot your computer in "safe mode." Choose safe mode by pressing F8 just before your machine starts Windows during re-booting, install the Connector/ODBC drivers, and re-boot to normal mode.

26.1.2.2. Installing Connector/ODBC from a Binary Distribution on Unix

There are two methods available for installing Connector/ODBC on Unix from a binary distribution. For most Unix environments you will need to use the tarball distribution. For Linux systems, there is also an RPM distribution available.

26.1.2.2.1. Installing Connector/ODBC from a Binary Tarball Distribution

To install the driver from a tarball distribution (`.tar.gz` file), download the latest version of the driver for your operating system and follow these steps that demonstrate the process using the Linux version of the tarball:

```
shell> su root
shell> gunzip mysql-connector-odbc-3.51.11-i686-pc-linux.tar.gz
shell> tar xvf mysql-connector-odbc-3.51.11-i686-pc-linux.tar
```

```
shell> cd mysql-connector-odbc-3.51.11-i686-pc-linux
```

Read the installation instructions in the [INSTALL - BINARY](#) file and execute these commands.

```
shell> cp libmyodbc* /usr/local/lib
shell> cp odbc.ini /usr/local/etc
shell> export ODBCINI=/usr/local/etc/odbc.ini
```

Then proceed on to [Section 26.1.3.4, “Configuring a Connector/ODBC DSN on Unix”](#), to configure the DSN for Connector/ODBC. For more information, refer to the [INSTALL - BINARY](#) file that comes with your distribution.

26.1.2.2.2. Installing Connector/ODBC from an RPM Distribution

To install or upgrade Connector/ODBC from an RPM distribution on Linux, simply download the RPM distribution of the latest version of Connector/ODBC and follow the instructions below. Use `su root` to become `root`, then install the RPM file.

If you are installing for the first time:

```
shell> su root
shell> rpm -ivh mysql-connector-odbc-3.51.12.i386.rpm
```

If the driver exists, upgrade it like this:

```
shell> su root
shell> rpm -Uvh mysql-connector-odbc-3.51.12.i386.rpm
```

If there is any dependency error for MySQL client library, `libmysqlclient`, simply ignore it by supplying the `--nodeps` option, and then make sure the MySQL client shared library is in the path or set through `LD_LIBRARY_PATH`.

This installs the driver libraries and related documents to `/usr/local/lib` and `/usr/share/doc/MyODBC`, respectively. Proceed onto [Section 26.1.3.4, “Configuring a Connector/ODBC DSN on Unix”](#).

To **uninstall** the driver, become `root` and execute an `rpm` command:

```
shell> su root
shell> rpm -e mysql-connector-odbc
```

26.1.2.3. Installing Connector/ODBC from a Binary Distribution on Mac OS X

Mac OS X is based on the FreeBSD operating system, and you can normally use the MySQL network port for connecting to MySQL servers on other hosts. Installing the Connector/ODBC driver enables you to connect to MySQL databases on any platform through the ODBC interface. You should only need to install the Connector/ODBC driver when your application requires an ODBC interface. Applications that require or can use ODBC (and therefore the Connector/ODBC driver) include ColdFusion, Filemaker Pro, 4th Dimension and many other applications.

Mac OS X includes its own ODBC manager, based on the `iODBC` manager. Mac OS X includes an administration tool that provides easier administration of ODBC drivers and configuration, updating the underlying `iODBC` configuration files.

The method for installing Connector/ODBC on Mac OS X depends on the version on Connector/ODBC you are using. For Connector/ODBC 3.51.14 and later, the package is provided as a compress tar archive that you must manually install. For Connector/ODBC 3.51.13 and earlier the software was provided on a compressed disk image (`.dmg`) file and included an installer.

In either case, the driver is designed to work with the `iODBC` driver manager included with Mac OS X.

To install Connector/ODBC 3.51.14 and later:

1. Download the installation file. Note that versions are available for both PowerPC and Intel platforms.
2. Extract the archive:

```
$ tar zxf mysql-connector-odbc-3.51.16-osx10.4-x86-32bit.tar.gz
```

3. The directory created will contain two subdirectories, `lib` and `bin`. You need to copy these to a suitable location such as `/usr/local`:

```
$ cp bin/* /usr/local/bin  
$ cp lib/* /usr/local/lib
```

4. Finally, you must register the driver with iODBC using the [myodbc3i](#) tool you just installed:

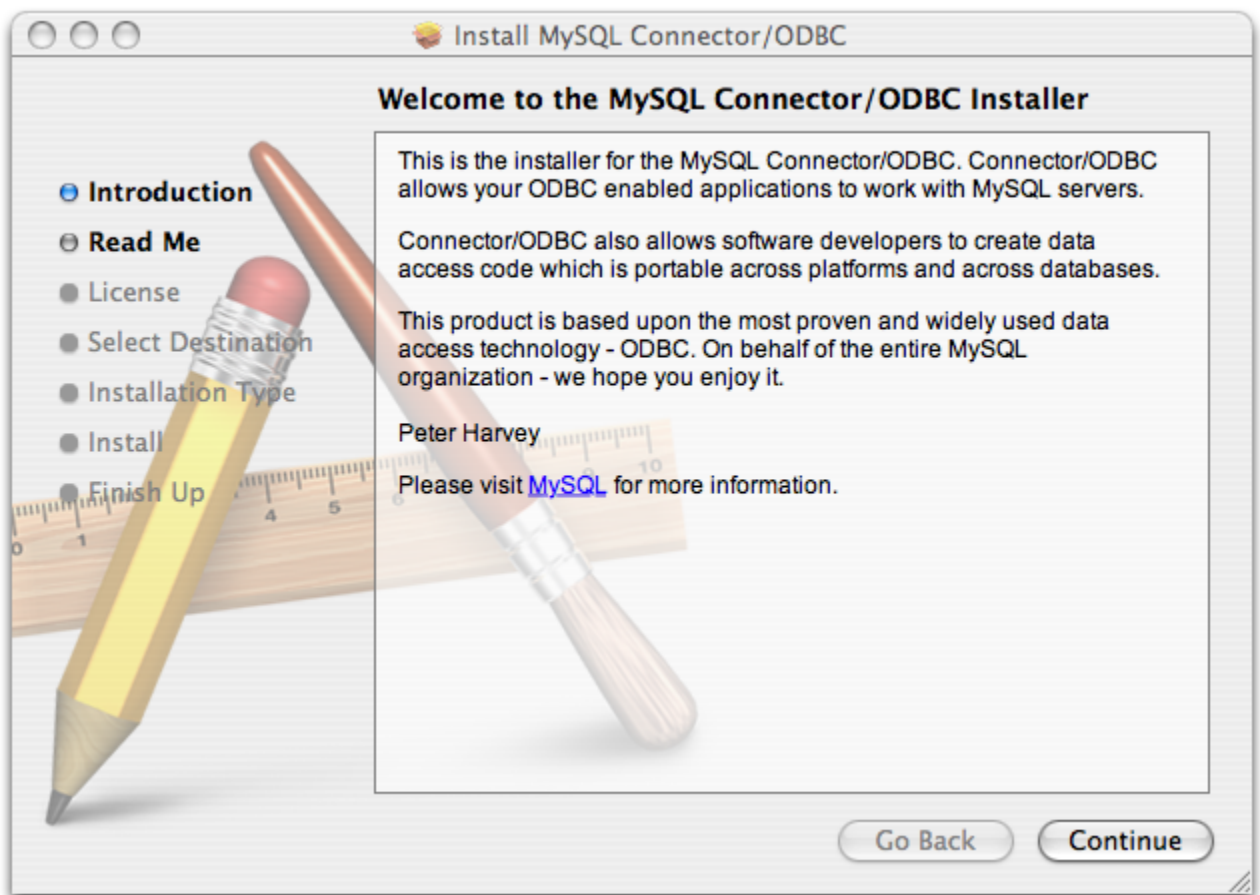
```
$ myodbc3i -a -d -t"MySQL ODBC 3.51 Driver;Driver=/usr/local/lib/libmyodbc3.so;Setup=/usr/local/lib/libmyodbc3S.so"
```

You can verify the installed drivers either by using the ODBC Administrator application or the [myodbc3i](#) utility:

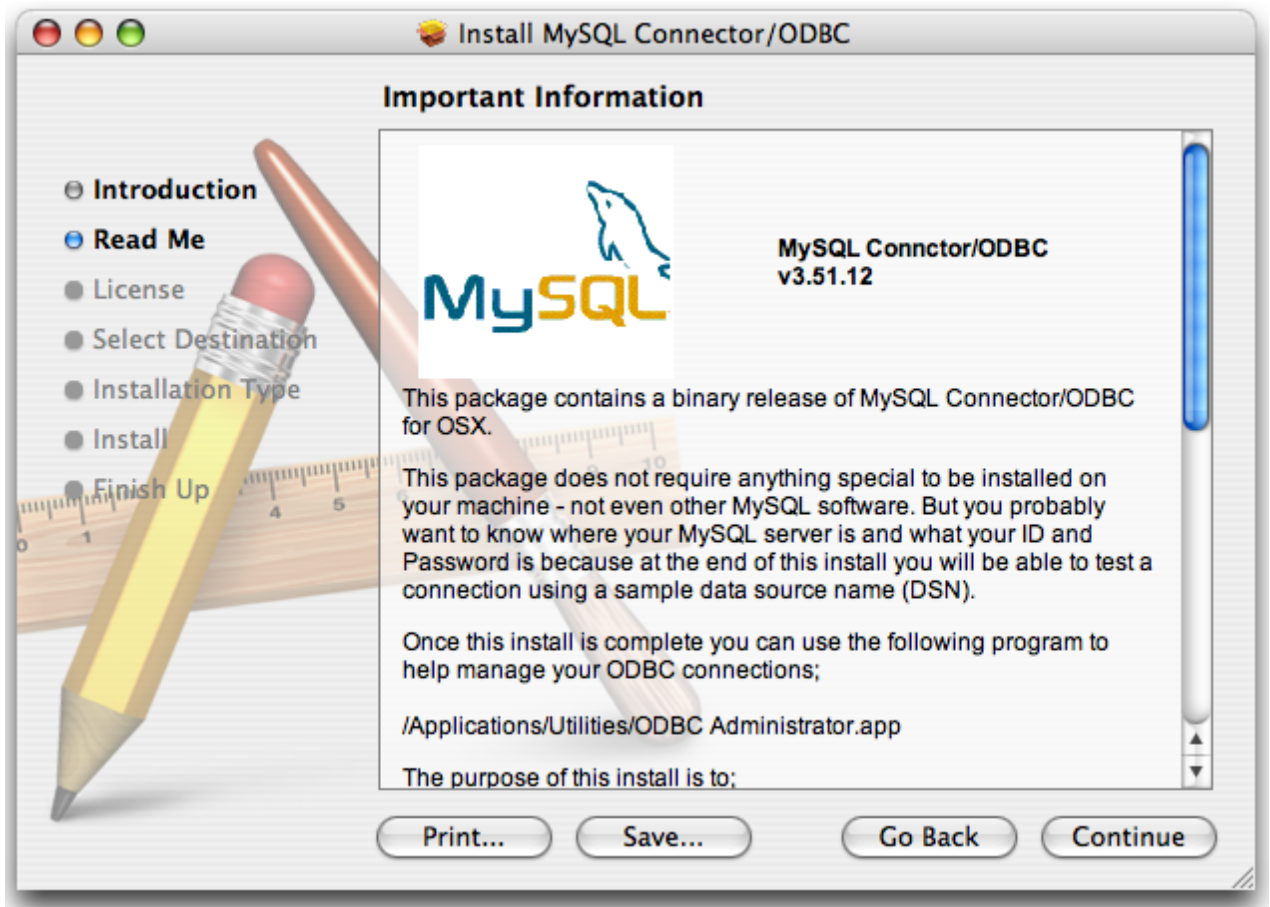
```
$ myodbc3i -q -d
```

To install Connector/ODBC 3.51.13 and earlier, follow these steps:

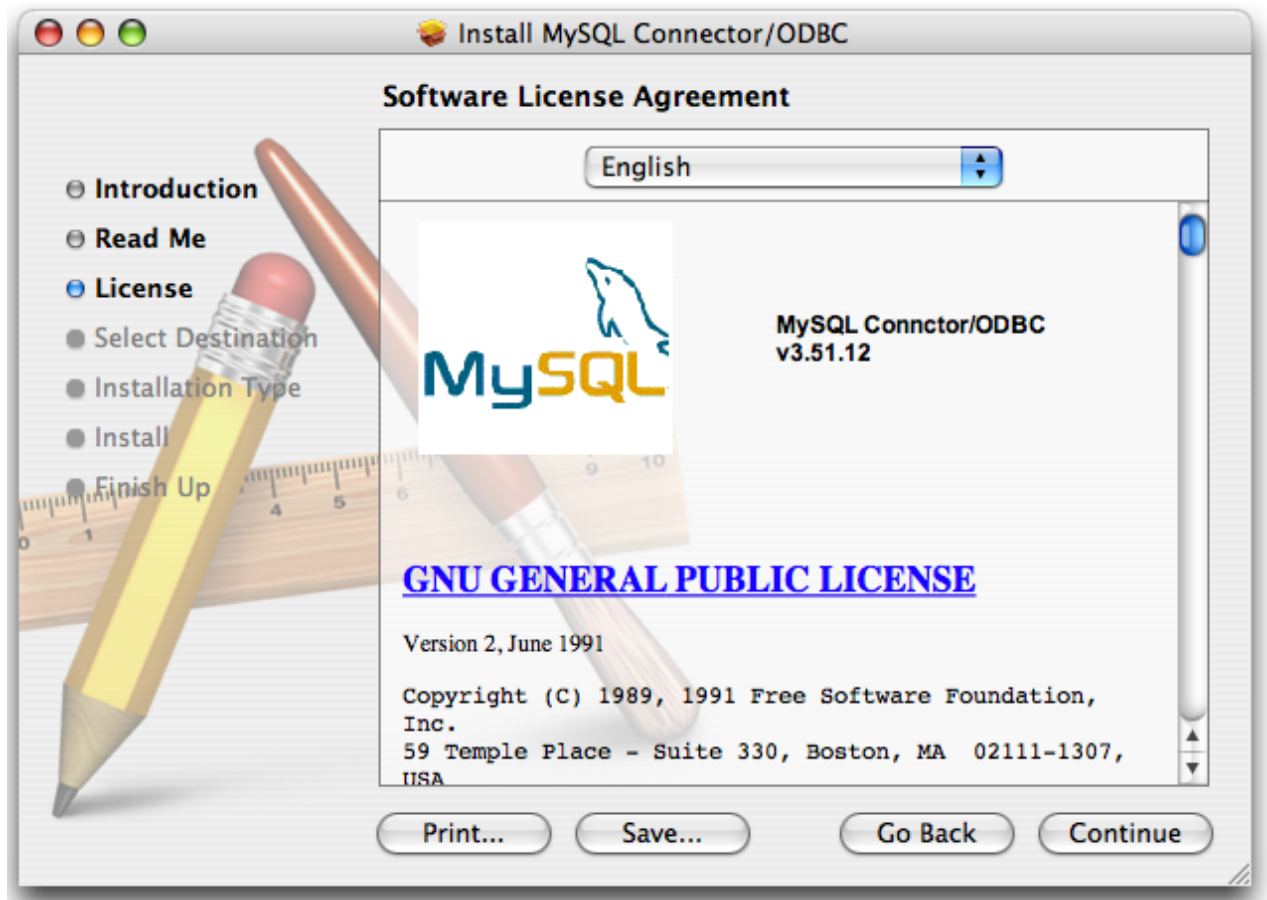
1. Download the file to your computer and double-click on the downloaded image file.
2. Within the disk image you will find an installer package (with the [.pkg](#) extension). Double click on this file to start the Mac OS X installer.
3. You will be presented with the installer welcome message. Click the CONTINUE button to begin the installation process.



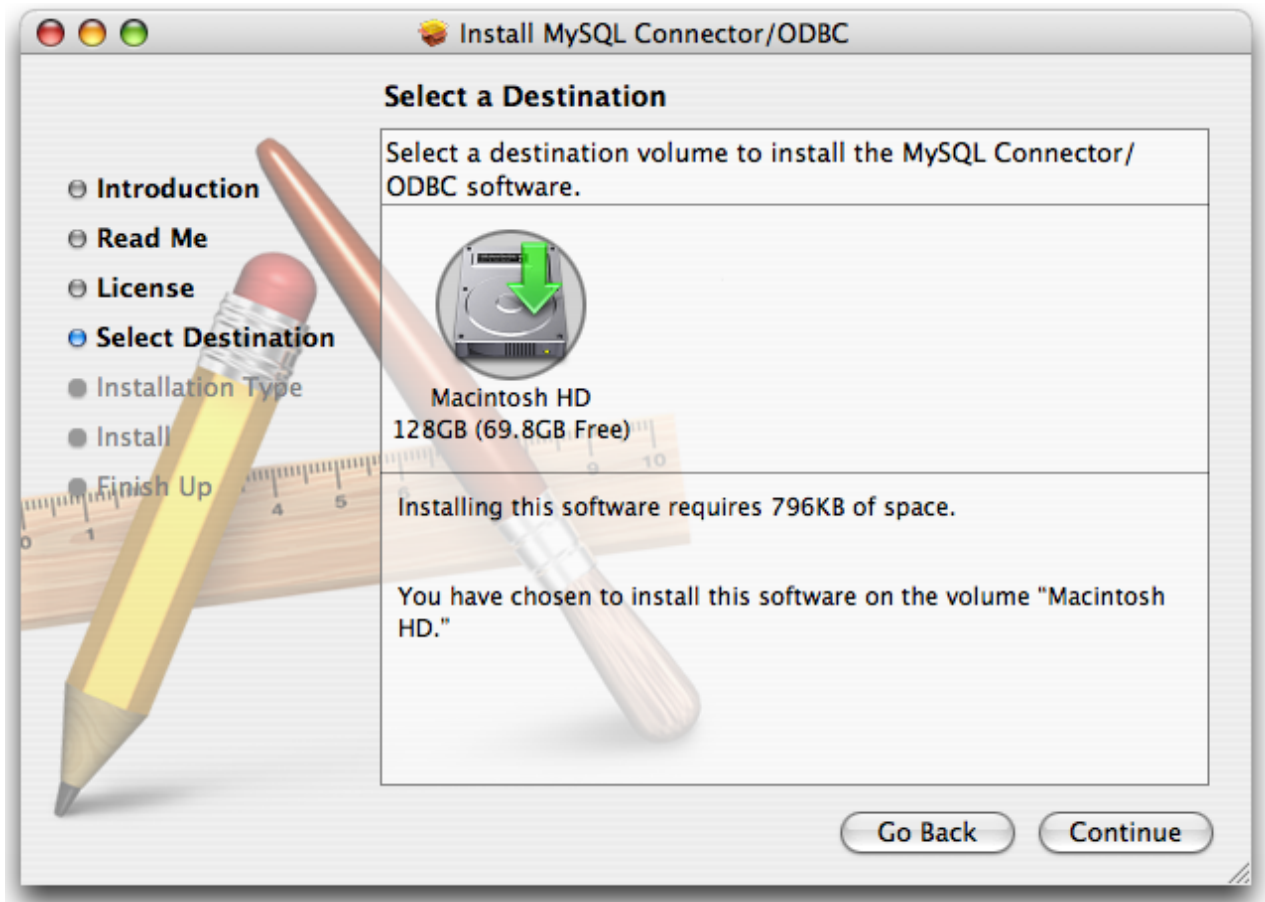
4. Please take the time to read the Important Information as it contains guidance on how to complete the installation process. Once you have read the notice and collected the necessary information, click CONTINUE.



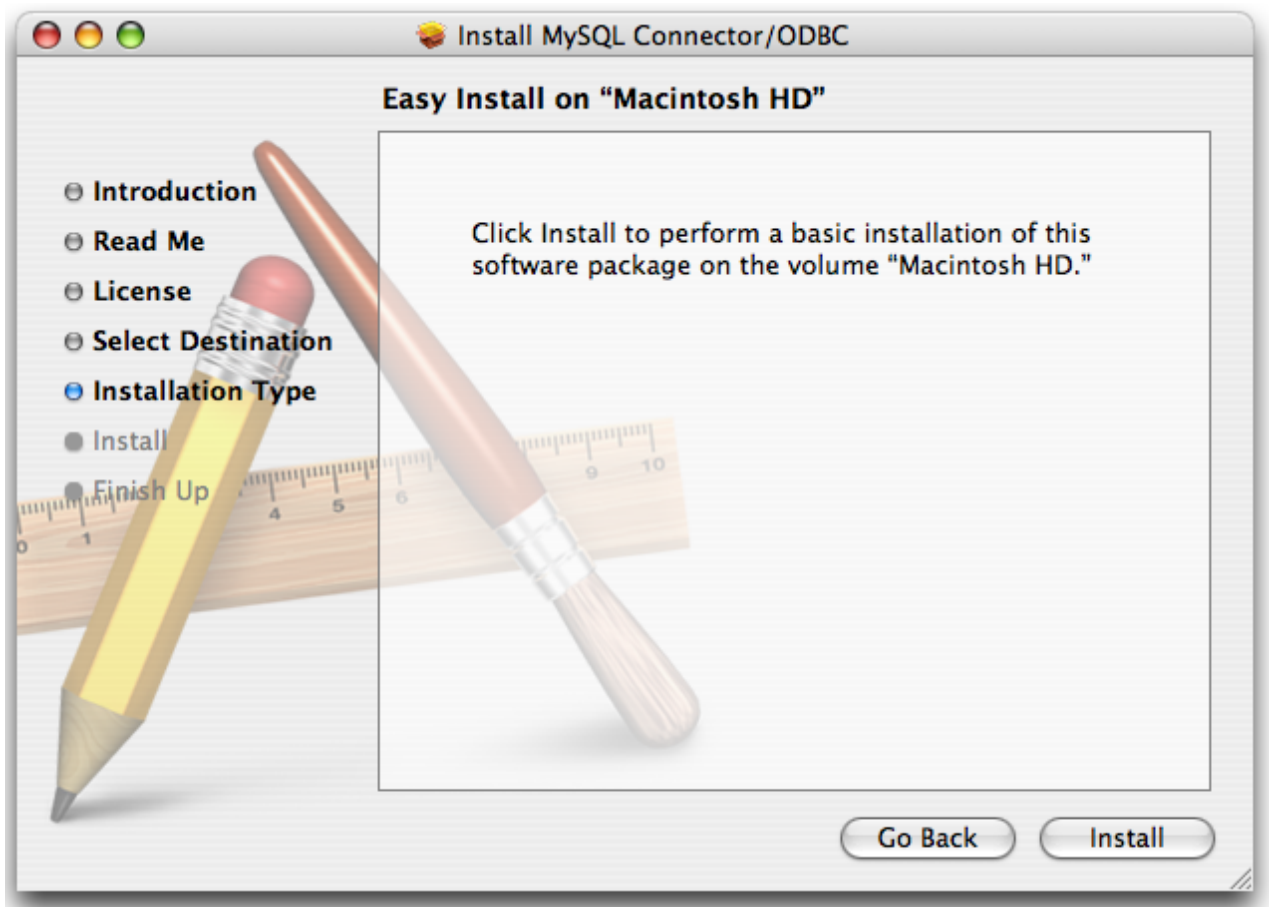
5. Connector/ODBC drivers are made available under the GNU General Public License. Please read the license if you are not familiar with it before continuing installation. Click CONTINUE to approve the license (you will be asked to confirm that decision) and continue the installation.



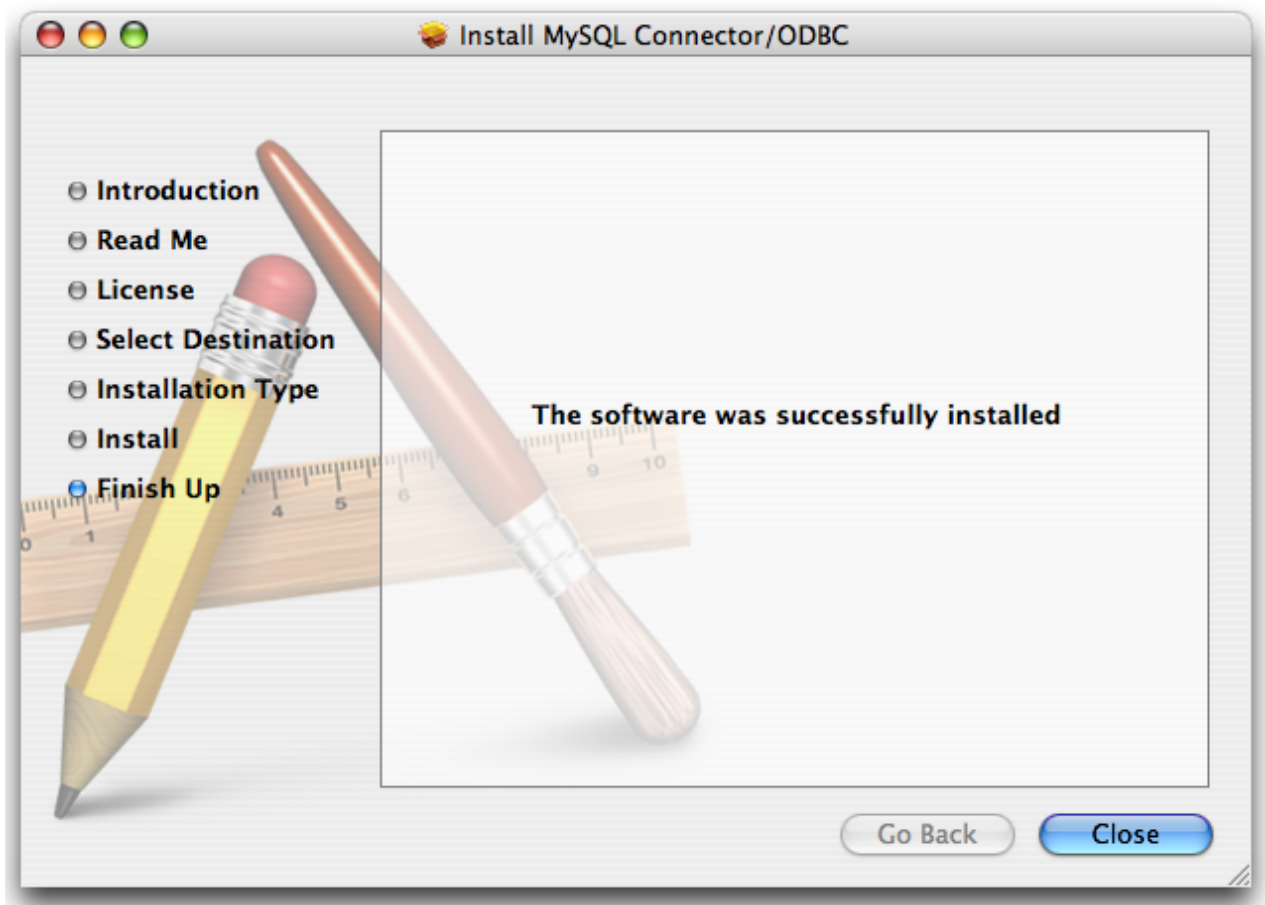
6. Choose a location to install the Connector/ODBC drivers and the ODBC Administrator application. You must install the files onto a drive with an operating system and you may be limited in the choices available. Select the drive you want to use, and then click CONTINUE.



7. The installer will automatically select the files that need to be installed on your machine. Click **INSTALL** to continue. The installer will copy the necessary files to your machine. A progress bar will be shown indicating the installation progress.



8. When installation has been completed you will get a window like the one shown below. Click CLOSE to close and quit the installer.



26.1.2.4. Installing Connector/ODBC from a Source Distribution on Windows

You should only need to install Connector/ODBC from source on Windows if you want to change or modify the source or installation. If you are unsure whether to install from source, please use the binary installation detailed in [Section 26.1.2.1, “Installing Connector/ODBC from a Binary Distribution on Windows”](#).

Installing Connector/ODBC from source on Windows requires a number of different tools and packages:

- MDAC, Microsoft Data Access SDK from <http://www.microsoft.com/data/>.
- Suitable C compiler, such as Microsoft Visual C++ or the C compiler included with Microsoft Visual Studio.
- Compatible `make` tool. Microsoft's `nmake` is used in the examples in this section.
- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because Connector/ODBC uses new calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit <http://dev.mysql.com/downloads/>.

26.1.2.4.1. Building Connector/ODBC 3.51

Connector/ODBC source distributions include `Makefiles` that require the `nmake` or other `make` utility. In the distribution, you can find `Makefile` for building the release version and `Makefile_debug` for building debugging versions of the driver libraries and DLLs.

To build the driver, use this procedure:

1. Download and extract the sources to a folder, then change directory into that folder. The following command assumes the folder is named `myodbc3-src`:

```
C:\> cd myodbc3-src
```

2. Edit `Makefile` to specify the correct path for the MySQL client libraries and header files. Then use the following commands to build and install the release version:

```
C:\> nmake -f Makefile
C:\> nmake -f Makefile install
```

`nmake -f Makefile` builds the release version of the driver and places the binaries in subdirectory called `Release`.

`nmake -f Makefile install` installs (copies) the driver DLLs and libraries (`myodbc3.dll`, `myodbc3.lib`) to your system directory.

3. To build the debug version, use `Makefile_Debug` rather than `Makefile`, as shown below:

```
C:\> nmake -f Makefile_debug
C:\> nmake -f Makefile_debug install
```

4. You can clean and rebuild the driver by using:

```
C:\> nmake -f Makefile clean
C:\> nmake -f Makefile install
```

Note

- Make sure to specify the correct MySQL client libraries and header files path in the Makefiles (set the `MYSQL_LIB_PATH` and `MYSQL_INCLUDE_PATH` variables). The default header file path is assumed to be `C:\mysql\include`. The default library path is assumed to be `C:\mysql\lib\opt` for release DLLs and `C:\mysql\lib\debug` for debug versions.
- For the complete usage of `nmake`, visit http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vcce4/html/evgrfRunningNMAKE.asp.
- If you are using the Subversion tree for compiling, all Windows-specific `Makefiles` are named as `Win_Makefile*`.

26.1.2.4.2. Testing

After the driver libraries are copied/installed to the system directory, you can test whether the libraries are properly built by using the samples provided in the `samples` subdirectory:

```
C:\> cd samples
C:\> nmake -f Makefile all
```

26.1.2.4.3. Building MyODBC 2.50

The MyODBC 2.50 source distribution includes VC workspace files. You can build the driver using these files (`.dsp` and `.dsw`) directly by loading them from Microsoft Visual Studio 6.0 or higher.

26.1.2.5. Installing Connector/ODBC from a Source Distribution on Unix

You need the following tools to build MySQL from source on Unix:

- A working ANSI C++ compiler. `gcc` 2.95.2 or later, `egcs` 1.0.2 or later or `egcs 2.91.66`, SGI C++, and SunPro C++ are some of the compilers that are known to work.
- A good `make` program. GNU `make` is always recommended and is sometimes required.

- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because Connector/ODBC uses new calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit <http://dev.mysql.com/downloads/>.

If you have built your own MySQL server and/or client libraries from source then you must have used the `-enable-thread-safe-client` option to `configure` when the libraries were built.

You should also ensure that the `libmysqlclient` library were built and installed as a shared library.

- A compatible ODBC manager must be installed. Connector/ODBC is known to work with the `iODBC` and `unixODBC` managers. See [Section 26.1.1.2.2, “ODBC Driver Managers”](#), for more information.
- If you are using a character set that isn't compiled into the MySQL client library then you need to install the MySQL character definitions from the `charsets` directory into `SHAREDIR` (by default, `/usr/local/mysql/share/mysql/charsets`). These should be in place if you have installed the MySQL server on the same machine. See [Chapter 9, Character Set Support](#), for more information on character set support.

Once you have all the required files, unpack the source files to a separate directory, you then have to run `configure` and build the library using `make`.

26.1.2.5.1. Typical `configure` Options

The `configure` script gives you a great deal of control over how you configure your Connector/ODBC build. Typically you do this using options on the `configure` command line. You can also affect `configure` using certain environment variables. For a list of options and environment variables supported by `configure`, run this command:

```
shell> ./configure --help
```

Some of the more commonly used `configure` options are described here:

- To compile Connector/ODBC, you need to supply the MySQL client include and library files path using the `-with-mysql-path=DIR` option, where `DIR` is the directory where MySQL is installed.

MySQL compile options can be determined by running `DIR/bin/mysql_config`.

- Supply the standard header and library files path for your ODBC Driver Manager (`iODBC` or `unixODBC`).
 - If you are using `iODBC` and `iODBC` is not installed in its default location (`/usr/local`), you might have to use the `-with-iodbc=DIR` option, where `DIR` is the directory where `iODBC` is installed.

If the `iODBC` headers do not reside in `DIR/include`, you can use the `--with-iodbc-includes=INCDIR` option to specify their location.

The applies to libraries. If they are not in `DIR/lib`, you can use the `--with-iodbc-libs=LIBDIR` option.

- If you are using `unixODBC`, use the `--with-unixODBC=DIR` option (case sensitive) to make `configure` look for `unixODBC` instead of `iODBC` by default, `DIR` is the directory where `unixODBC` is installed.
- If the `unixODBC` headers and libraries aren't located in `DIR/include` and `DIR/lib`, use the `-with-unixODBC-includes=INCDIR` and `--with-unixODBC-libs=LIBDIR` options.
- You might want to specify an installation prefix other than `/usr/local`. For example, to install the Connector/ODBC drivers in `/usr/local/odbc/lib`, use the `--prefix=/usr/local/odbc` option.

The final configuration command looks something like this:

```
shell> ./configure --prefix=/usr/local \
  --with-iodbc=/usr/local \
  --with-mysql-path=/usr/local/mysql
```

26.1.2.5.2. Additional `configure` Options

There are a number of other options that you need, or want, to set when configuring the Connector/ODBC driver before it is built.

- To link the driver with MySQL thread safe client libraries `libmysqlclient_r.so` or `libmysqlclient_r.a`, you must specify the following `configure` option:

```
--enable-thread-safe
```

and can be disabled (default) using

```
--disable-thread-safe
```

This option enables the building of the driver thread-safe library `libmyodbc3_r.so` from by linking with MySQL thread-safe client library `libmysqlclient_r.so` (The extensions are OS dependent).

If the compilation with the thread-safe option fails, it may be because the correct thread-libraries on the system could not be located. You should set the value of `LIBS` to point to the correct thread library for your system.

```
LIBS="-lpthread" ./configure ..
```

- You can enable or disable the shared and static versions of Connector/ODBC using these options:

```
--enable-shared[=yes/no]
--disable-shared
--enable-static[=yes/no]
--disable-static
```

- By default, all the binary distributions are built as non-debugging versions (configured with `--without-debug`).

To enable debugging information, build the driver from source distribution and use the `--with-debug` option when you run `configure`.

- This option is available only for source trees that have been obtained from the Subversion repository. This option does not apply to the packaged source distributions.

By default, the driver is built with the `--without-docs` option. If you would like the documentation to be built, then execute `configure` with:

```
--with-docs
```

26.1.2.5.3. Building and Compilation

To build the driver libraries, you have to just execute `make`.

```
shell> make
```

If any errors occur, correct them and continue the build process. If you aren't able to build, then send a detailed email to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com) for further assistance.

26.1.2.5.4. Building Shared Libraries

On most platforms, MySQL does not build or support `.so` (shared) client libraries by default. This is based on our experience of problems when building shared libraries.

In cases like this, you have to download the MySQL distribution and configure it with these options:

```
--without-server --enable-shared
```

To build shared driver libraries, you must specify the `--enable-shared` option for `configure`. By default, `configure` does not enable this option.

If you have configured with the `--disable-shared` option, you can build the `.so` file from the static libraries using the following

commands:

```
shell> cd mysql-connector-odbc-3.51.01
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
    $CC -bundle -flat_namespace -undefined error \
    -o .libs/libmyodbc3-3.51.01.so \
    catalog.o connect.o cursor.o dll.o error.o execute.o \
    handle.o info.o misc.o myodbc3.o options.o prepare.o \
    results.o transact.o utility.o \
    -L/usr/local/mysql/lib/mysql/ \
    -L/usr/local/iodbc/lib/ \
    -lz -lc -lmysqlclient -liodbcinst
```

Make sure to change `-liodbcinst` to `-lodbcinst` if you are using `unixODBC` instead of `iODBC`, and configure the library paths accordingly.

This builds and places the `libmyodbc3-3.51.01.so` file in the `.libs` directory. Copy this file to the Connector/ODBC library installation directory (`/usr/local/lib` (or the `lib` directory under the installation directory that you supplied with the `--prefix`).

```
shell> cd .libs
shell> cp libmyodbc3-3.51.01.so /usr/local/lib
shell> cd /usr/local/lib
shell> ln -s libmyodbc3-3.51.01.so libmyodbc3.so
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
    $CC -bundle -flat_namespace -undefined error \
    -o .libs/libmyodbc3_r-3.51.01.so \
    catalog.o connect.o cursor.o dll.o error.o execute.o \
    handle.o info.o misc.o myodbc3.o options.o prepare.o \
    results.o transact.o utility.o \
    -L/usr/local/mysql/lib/mysql/ \
    -L/usr/local/iodbc/lib/ \
    -lz -lc -lmysqlclient_r -liodbcinst
```

26.1.2.5.5. Installing Driver Libraries

To install the driver libraries, execute the following command:

```
shell> make install
```

That command installs one of the following sets of libraries:

For Connector/ODBC 3.51:

- `libmyodbc3.so`
- `libmyodbc3-3.51.01.so`, where 3.51.01 is the version of the driver
- `libmyodbc3.a`

For thread-safe Connector/ODBC 3.51:

- `libmyodbc3_r.so`
- `libmyodbc3-3_r.51.01.so`
- `libmyodbc3_r.a`

For MyODBC 2.5.0:

- `libmyodbc.so`

- `libmyodbc-2.50.39.so`, where 2.50.39 is the version of the driver
- `libmyodbc.a`

For more information on build process, refer to the [INSTALL](#) file that comes with the source distribution. Note that if you are trying to use the [make](#) from Sun, you may end up with errors. On the other hand, GNU [gmake](#) should work fine on all platforms.

26.1.2.5.6. Testing Connector/ODBC on Unix

To run the basic samples provided in the distribution with the libraries that you built, use the following command:

```
shell> make test
```

Before running the tests, create the DSN 'myodbc3' in `odbc.ini` and set the environment variable `ODBCINI` to the correct `odbc.ini` file; and MySQL server is running. You can find a sample `odbc.ini` with the driver distribution.

You can even modify the `samples/run-samples` script to pass the desired DSN, UID, and PASSWORD values as the command-line arguments to each sample.

26.1.2.5.7. Building Connector/ODBC from Source on Mac OS X

To build the driver on Mac OS X (Darwin), make use of the following `configure` example:

```
shell> ./configure --prefix=/usr/local
--with-unixODBC=/usr/local
--with-mysql-path=/usr/local/mysql
--disable-shared
--enable-gui=no
--host=powerpc-apple
```

The command assumes that the `unixODBC` and MySQL are installed in the default locations. If not, configure accordingly.

On Mac OS X, `--enable-shared` builds `.dylib` files by default. You can build `.so` files like this:

```
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
$CC -bundle -flat_namespace -undefined error
-o .libs/libmyodbc3-3.51.01.so *.o
-L/usr/local/mysql/lib/
-L/usr/local/iodbc/lib
-liodbcinst -lmysqlclient -lz -lc
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
$CC -bundle -flat_namespace -undefined error
-o .libs/libmyodbc3-3.51.01.so *.o
-L/usr/local/mysql/lib/
-L/usr/local/iodbc/lib
-liodbcinst -lmysqlclient_r -lz -lc -lpthread
```

Make sure to change the `-liodbcinst` to `-lodbcinst` in case of using `unixODBC` instead of `iodbc` and configure the libraries path accordingly.

In Apple's version of GCC, both `cc` and `gcc` are actually symbolic links to `gcc3`.

Copy this library to the `$prefix/lib` directory and symlink to `libmyodbc3.so`.

You can cross-check the output shared-library properties using this command:

```
shell> otool -LD .libs/libmyodbc3-3.51.01.so
```

26.1.2.5.8. Building Connector/ODBC from Source on HP-UX

To build the driver on HP-UX 10.x or 11.x, make use of the following `configure` example:

If using `CC`:

```
shell> CC="cc" \
      CFLAGS="+z" \
      LDFLAGS="-Wl,+b:-Wl,+s" \
      ./configure --prefix=/usr/local
      --with-unixodbc=/usr/local
      --with-mysql-path=/usr/local/mysql/lib/mysql
      --enable-shared
      --enable-thread-safe
```

If using `gcc`:

```
shell> CC="gcc" \
      LDFLAGS="-Wl,+b:-Wl,+s" \
      ./configure --prefix=/usr/local
      --with-unixodbc=/usr/local
      --with-mysql-path=/usr/local/mysql
      --enable-shared
      --enable-thread-safe
```

Once the driver is built, cross-check its attributes using `chatr .libs/libmyodbc3.sl` to determine whether you need to have set the MySQL client library path using the `SHLIB_PATH` environment variable. For static versions, ignore all shared-library options and run `configure` with the `--disable-shared` option.

26.1.2.5.9. Building Connector/ODBC from Source on AIX

To build the driver on AIX, make use of the following `configure` example:

```
shell> ./configure --prefix=/usr/local
      --with-unixodbc=/usr/local
      --with-mysql-path=/usr/local/mysql
      --disable-shared
      --enable-thread-safe
```

Note

For more information about how to build and set up the static and shared libraries across the different platforms refer to '[Using static and shared libraries across platforms](#)'.

26.1.2.6. Installing Connector/ODBC from the Development Source Tree

Caution

You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL Connector/ODBC up and running on your system, you should use a standard release distribution.

To be able to access the Connector/ODBC source tree, you must have Subversion installed. Subversion is freely available from <http://subversion.tigris.org/>.

To build from the source trees, you need the following tools:

- autoconf 2.52 (or newer)
- automake 1.4 (or newer)
- libtool 1.4 (or newer)
- m4

The most recent development source tree is available from our public Subversion trees at <http://dev.mysql.com/tech-resources/sources.html>.

To checkout out the Connector/ODBC sources, change to the directory where you want the copy of the Connector/ODBC tree to be stored, then use the following command:

```
shell> svn co http://svn.mysql.com/svnpublic/connector-odbc3
```

You should now have a copy of the entire Connector/ODBC source tree in the directory `connector-odbc3`. To build from this source tree on Unix or Linux follow these steps:

```
shell> cd connector-odbc3
shell> aclocal
shell> autoheader
shell> autoconf
shell> automake;
shell> ./configure # Add your favorite options here
shell> make
```

For more information on how to build, refer to the `INSTALL` file located in the same directory. For more information on options to `configure`, see [Section 26.1.2.5.1, “Typical configure Options”](#)

When the build is done, run `make install` to install the Connector/ODBC 3.51 driver on your system.

If you have gotten to the `make` stage and the distribution does not compile, please report it to <myodbc@lists.mysql.com>.

On Windows, make use of Windows Makefiles `WIN-Makefile` and `WIN-Makefile_debug` in building the driver. For more information, see [Section 26.1.2.4, “Installing Connector/ODBC from a Source Distribution on Windows”](#).

After the initial checkout operation to get the source tree, you should run `svn update` periodically update your source according to the latest version.

26.1.3. Connector/ODBC Configuration

Before you connect to a MySQL database using the Connector/ODBC driver you must configure an ODBC *Data Source Name*. The DSN associates the various configuration parameters required to communicate with a database to a specific name. You use the DSN in an application to communicate with the database, rather than specifying individual parameters within the application itself. DSN information can be user specific, system specific, or provided in a special file. ODBC data source names are configured in different ways, depending on your platform and ODBC driver.

26.1.3.1. Data Source Names

A Data Source Name associates the configuration parameters for communicating with a specific database. Generally a DSN consists of the following parameters:

- Name
- Hostname
- Database Name
- Login
- Password

In addition, different ODBC drivers, including Connector/ODBC, may accept additional driver-specific options and parameters.

There are three types of DSN:

- A *System DSN* is a global DSN definition that is available to any user and application on a particular system. A System DSN can normally only be configured by a systems administrator, or by a user who has specific permissions that let them create System DSNs.
- A *User DSN* is specific to an individual user, and can be used to store database connectivity information that the user regularly uses.
- A *File DSN* uses a simple file to define the DSN configuration. File DSNs can be shared between users and machines and are therefore more practical when installing or deploying DSN information as part of an application across many machines.

DSN information is stored in different locations depending on your platform and environment.

26.1.3.2. Configuring a Connector/ODBC DSN on Windows

The **ODBC Data Source Administrator** within Windows enables you to create DSNs, check driver installation and configure ODBC systems such as tracing (used for debugging) and connection pooling.

Different editions and versions of Windows store the **ODBC Data Source Administrator** in different locations depending on the version of Windows that you are using.

To open the **ODBC Data Source Administrator** in Windows Server 2003:

1. On the **Start** menu, choose **Administrative Tools**, and then click **Data Sources (ODBC)**.

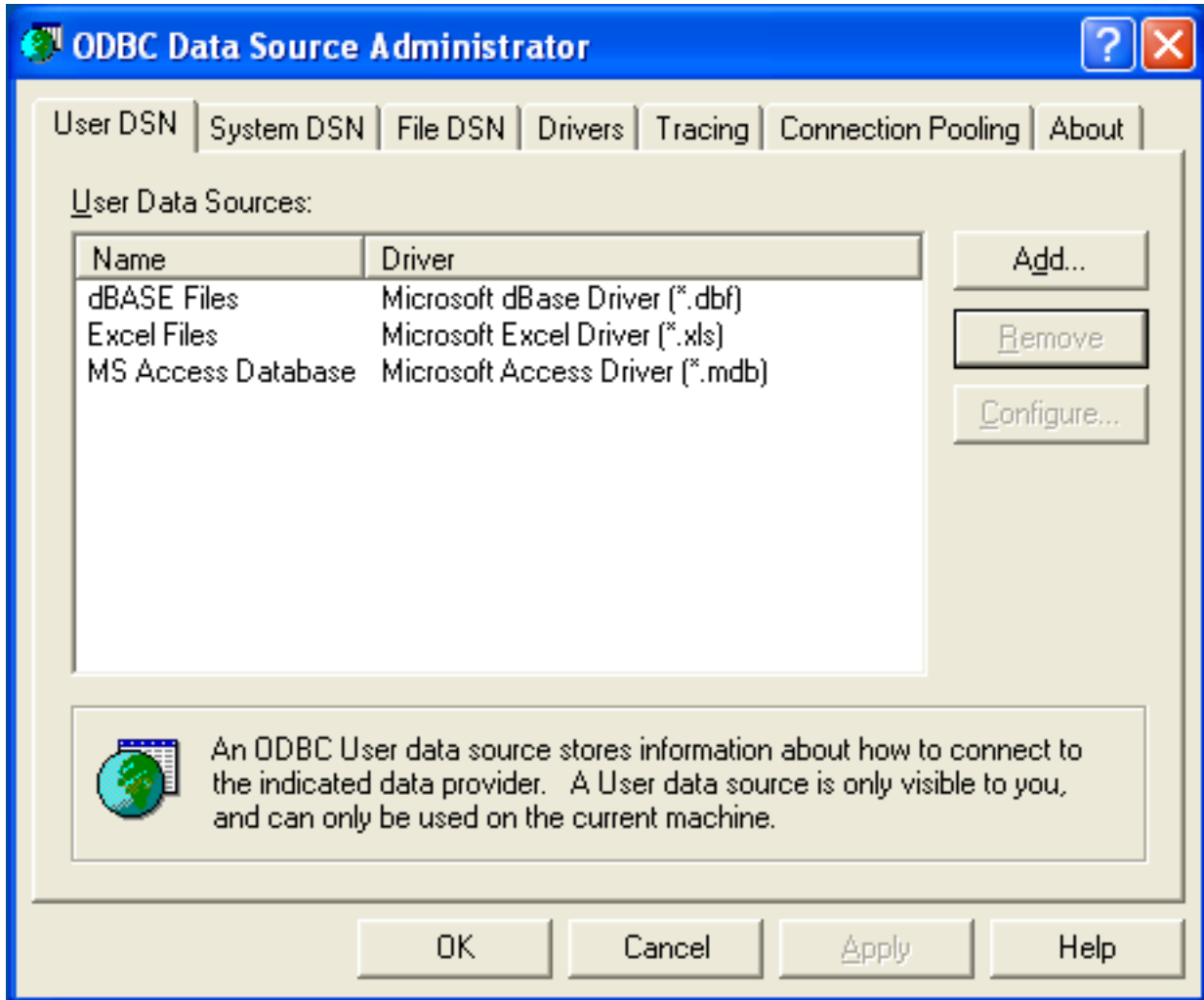
To open the **ODBC Data Source Administrator** in Windows 2000 Server or Windows 2000 Professional:

1. On the **Start** menu, choose **Settings**, and then click **Control Panel**.
2. In **Control Panel**, click **Administrative Tools**.
3. In **Administrative Tools**, click **Data Sources (ODBC)**.

To open the **ODBC Data Source Administrator** on Windows XP:

1. On the **Start** menu, click **Control Panel**.
2. In the **Control Panel** when in **Category View** click **Performance and Maintenance** and then click **Administrative Tools**.. If you are viewing the **Control Panel** in **Classic View**, click **Administrative Tools**.
3. In **Administrative Tools**, click **Data Sources (ODBC)**.

Irrespective of your Windows version, you should be presented the **ODBC Data Source Administrator** window:



Within Windows XP, you can add the [Administrative Tools](#) folder to your START menu to make it easier to locate the ODBC Data Source Administrator. To do this:

1. Right click on the START menu.
2. Select [Properties](#).
3. Click CUSTOMIZE....
4. Select the ADVANCED tab.
5. Within [Start menu items](#), within the [System Administrative Tools](#) section, select [Display on the All Programs menu](#).

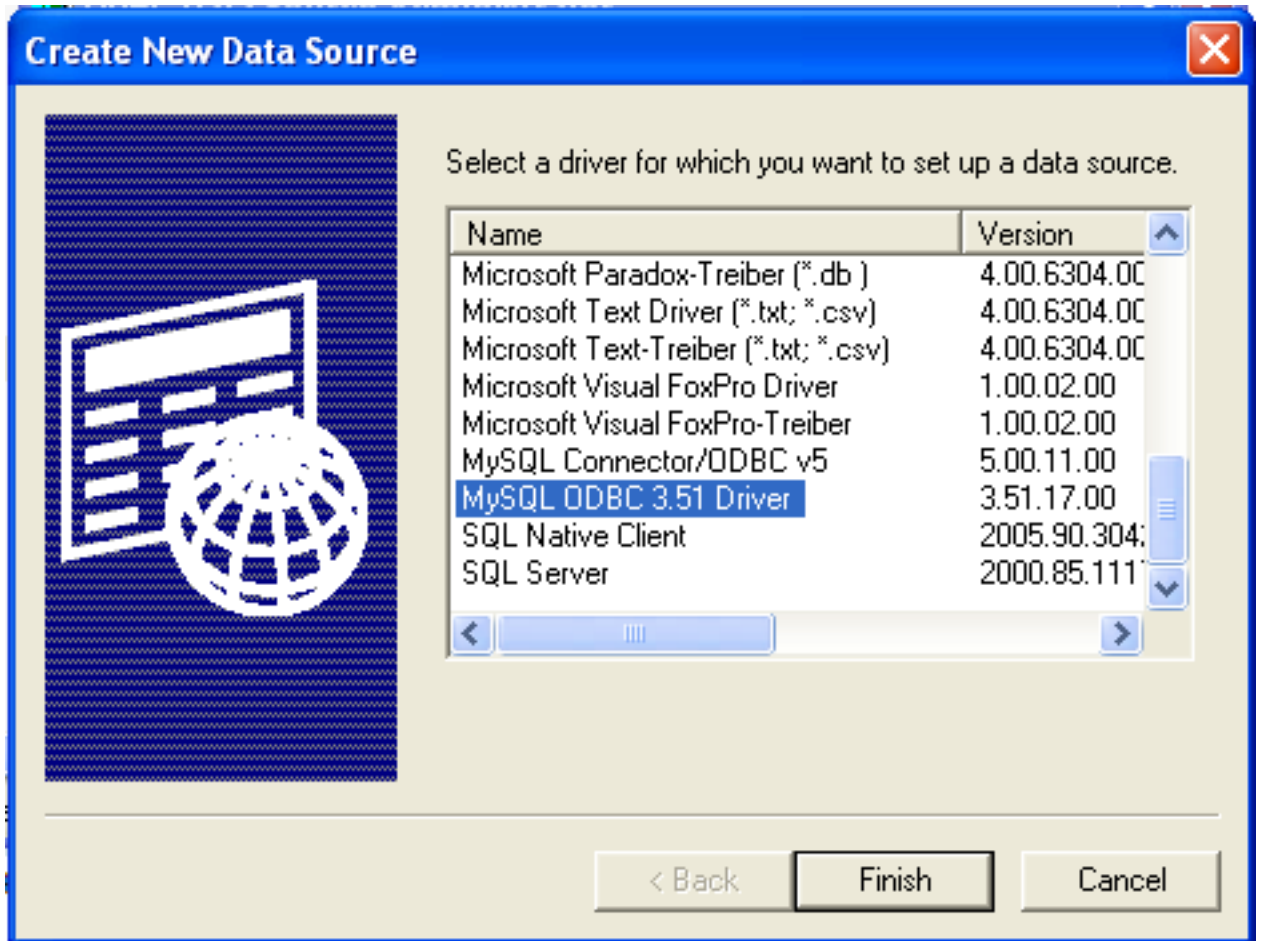
Within both Windows Server 2003 and Windows XP you may want to permanently add the [ODBC Data Source Administrator](#) to your START menu. To do this, locate the [Data Sources \(ODBC\)](#) icon using the methods shown, then right-click on the icon and then choose PIN TO START MENU.

26.1.3.2.1. Adding a Connector/ODBC DSN on Windows

To add and configure a new Connector/ODBC data source on Windows, use the [ODBC Data Source Administrator](#):

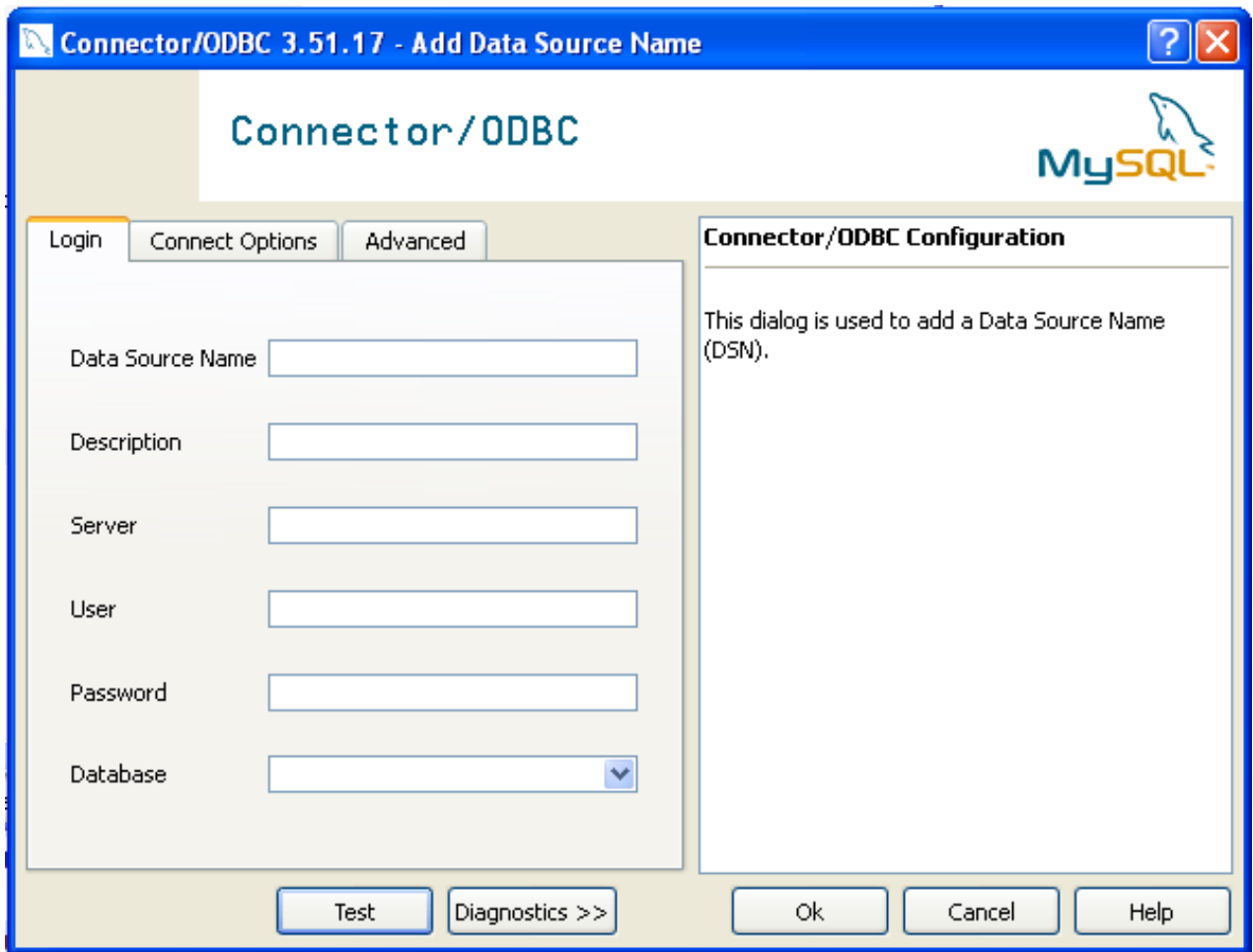
1. Open the [ODBC Data Source Administrator](#).

2. To create a System DSN (which will be available to all users) , select the [System DSN](#) tab. To create a User DSN, which will be unique only to the current user, click the [ADD...](#) button.
3. You will need to select the ODBC driver for this DSN.



Select [MySQL ODBC 3.51 Driver](#), then click [Finish](#).

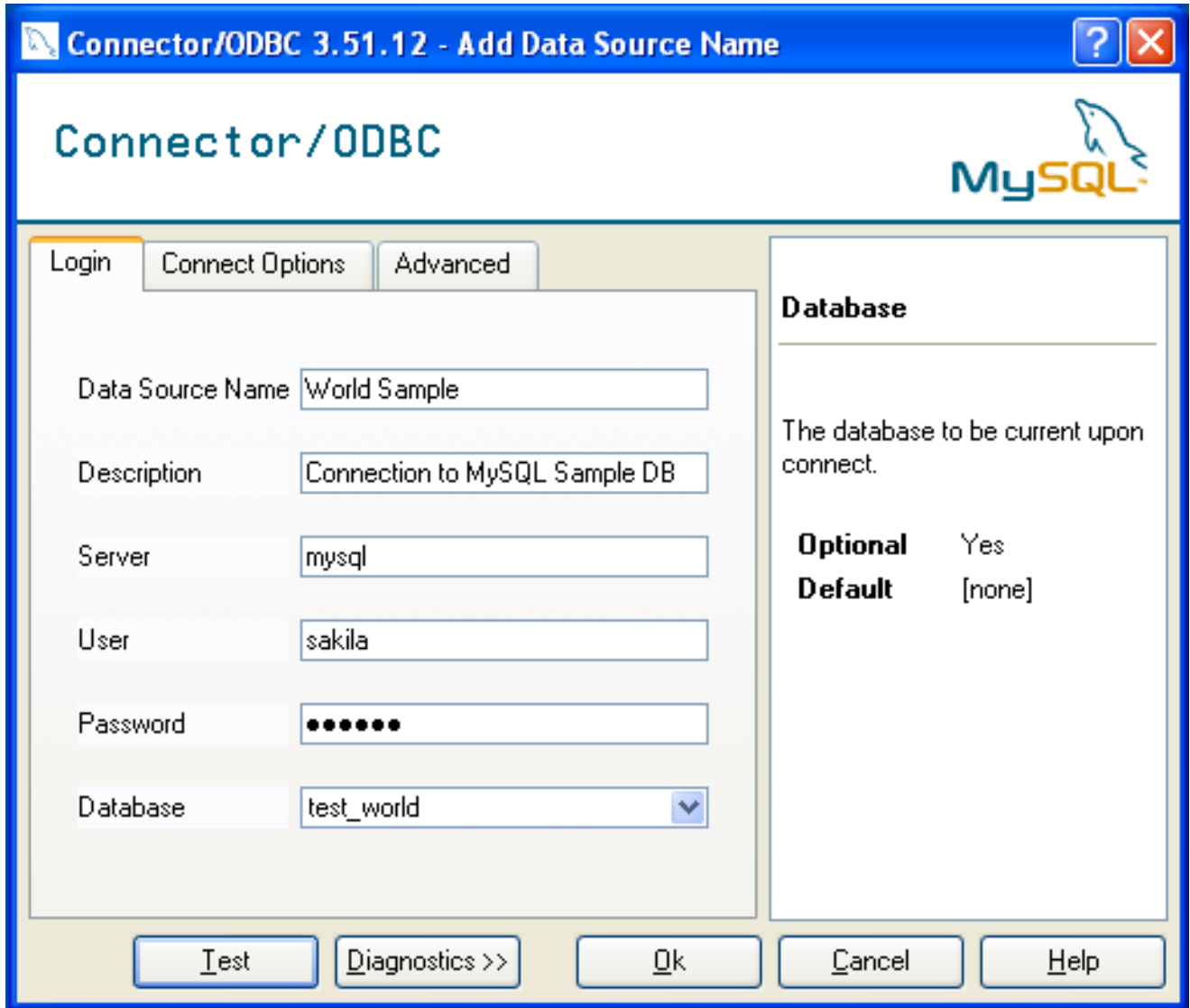
4. You now need to configure the specific fields for the DSN you are creating through the [Add Data Source Name](#) dialog.



In the **Data Source Name** box, enter the name of the data source you want to access. It can be any valid name that you choose.

5. In the **Description** box, enter some text to help identify the connection.
6. In the **Server** field, enter the name of the MySQL server host that you want to access. By default, it is `localhost`.
7. In the **User** field, enter the user name to use for this connection.
8. In the **Password** field, enter the corresponding password for this connection.
9. The **Database** popup should automatically populate with the list of databases that the user has permissions to access.
10. Click OK to save the DSN.

A completed DSN configuration may look like this:



26.1.3.2.2. Checking Connector/ODBC DSN Configuration on Windows

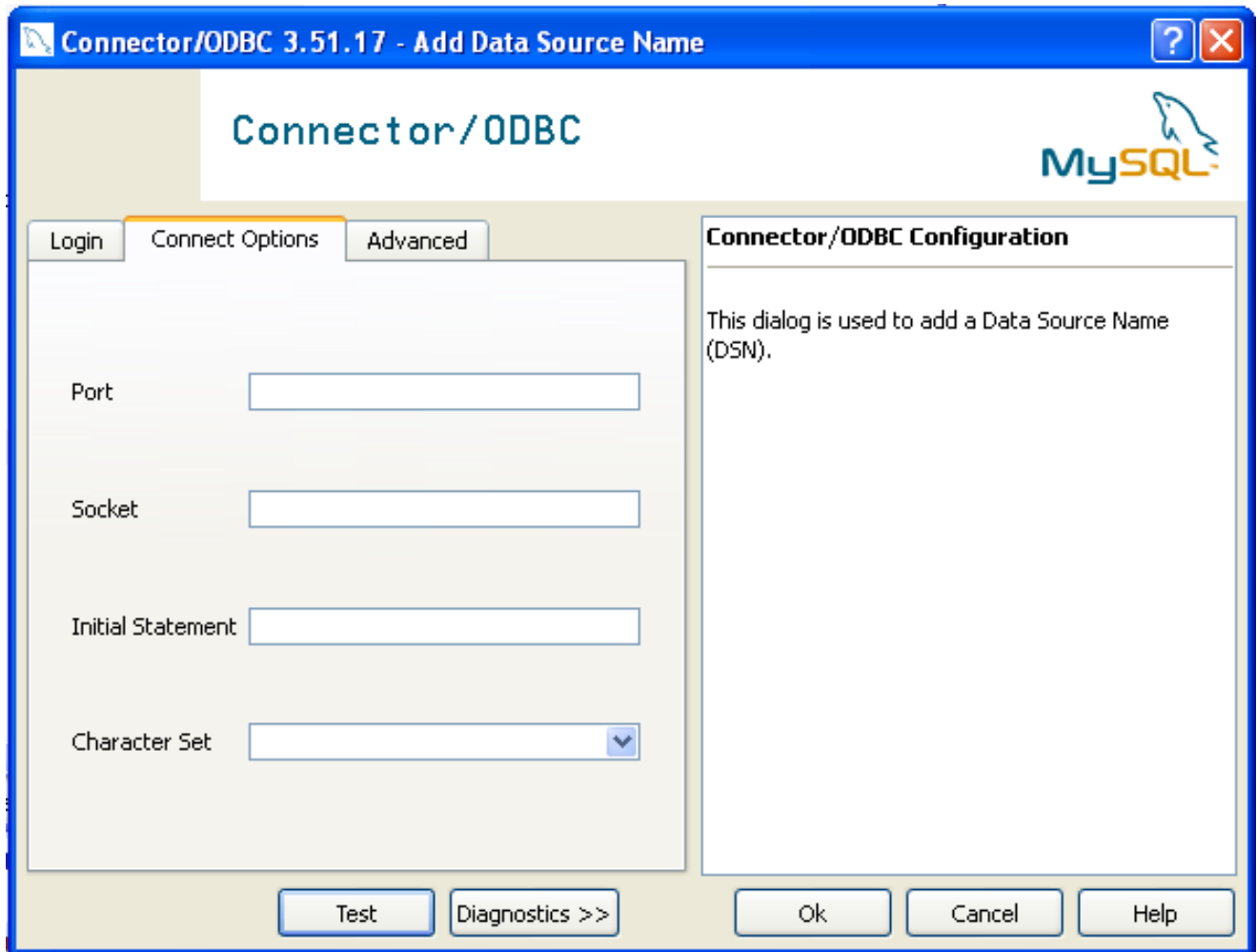
You can verify the connection using the parameters you have entered by clicking the TEST button. If the connection could be made successfully, you will be notified with a **Success; connection was made!** dialog.

If the connection failed, you can obtain more information on the test and why it may have failed by clicking the DIAGNOSTICS... button to show additional error messages.

26.1.3.2.3. Connector/ODBC DSN Configuration Options

You can configure a number of options for a specific DSN by using either the CONNECT OPTIONS or ADVANCED tabs in the DSN configuration dialog.

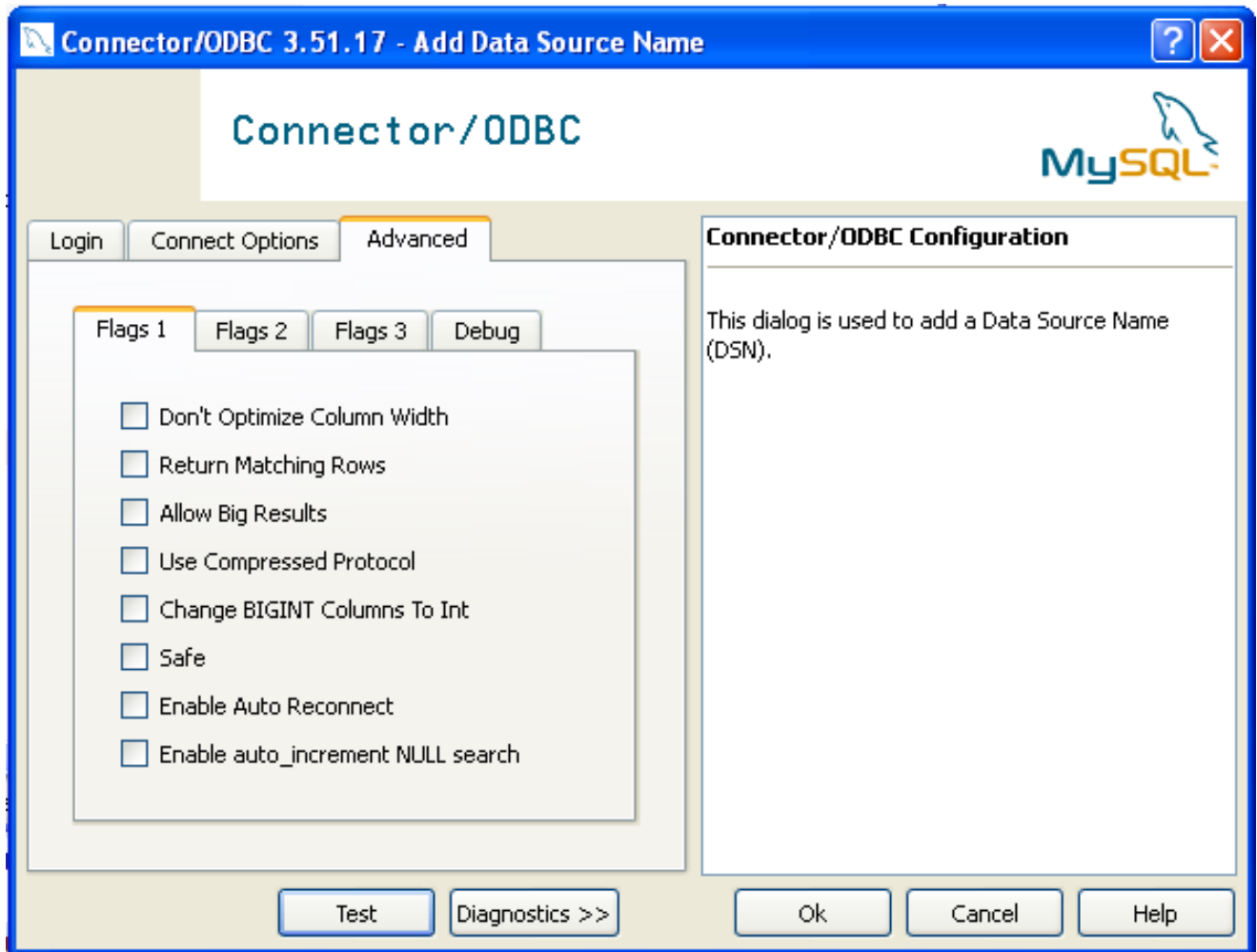
The CONNECTION OPTIONS dialog can be seen below.



The three options you can configure are:

- **Port** sets the TCP/IP port number to use when communicating with MySQL. Communication with MySQL uses port 3306 by default. If your server is configured to use a different TCP/IP port, you must specify that port number here.
- **Socket** sets the name or location of a specific socket or Windows pipe to use when communicating with MySQL.
- **Initial Statement** defines an SQL statement that will be executed when the connection to MySQL is opened. You can use this to set MySQL options for your connection, such as disabling autocommit.
- **Character Set** is a popup list from which you can select the default character set to be used with this connection. The Character Set option was added in 3.5.17.

The **ADVANCED** tab enables you to configure Connector/ODBC connection parameters. Refer to [Section 26.1.3.5, “Connector/ODBC Connection Parameters”](#), for information about the meaning of these options.



26.1.3.2.4. Errors and Debugging

This section answers Connector/ODBC connection-related questions.

- **While configuring a Connector/ODBC DSN, a `Could Not Load Translator or Setup Library` error occurs**

For more information, refer to [MS KnowledgeBase Article\(Q260558\)](#). Also, make sure you have the latest valid `ct13d32.dll` in your system directory.

- On Windows, the default `myodbc3.dll` is compiled for optimal performance. If you want to debug Connector/ODBC 3.51 (for example, to enable tracing), you should instead use `myodbc3d.dll`. To install this file, copy `myodbc3d.dll` over the installed `myodbc3.dll` file. Make sure to revert back to the release version of the driver DLL once you are done with the debugging because the debug version may cause performance issues. Note that the `myodbc3d.dll` isn't included in Connector/ODBC 3.51.07 through 3.51.11. If you are using one of these versions, you should copy that DLL from a previous version (for example, 3.51.06).

For MyODBC 2.50, `myodbc.dll` and `myodbcd.dll` are used instead.

26.1.3.3. Configuring a Connector/ODBC DSN on Mac OS X

To configure a DSN on Mac OS X you can either use the `myodbc3i` utility, edit the `odbc.ini` file within the `Library/ODBC` directory of the user or the should use the ODBC Administrator. If you have Mac OS X 10.2 or earlier, refer to [Section 26.1.3.4, “Configuring a Connector/ODBC DSN on Unix”](#). Select whether you want to create a User DSN or a System DSN. If you want to add a System DSN, you may need to authenticate with the system. You must click the padlock and enter a user and password with administrator privileges.

Warning

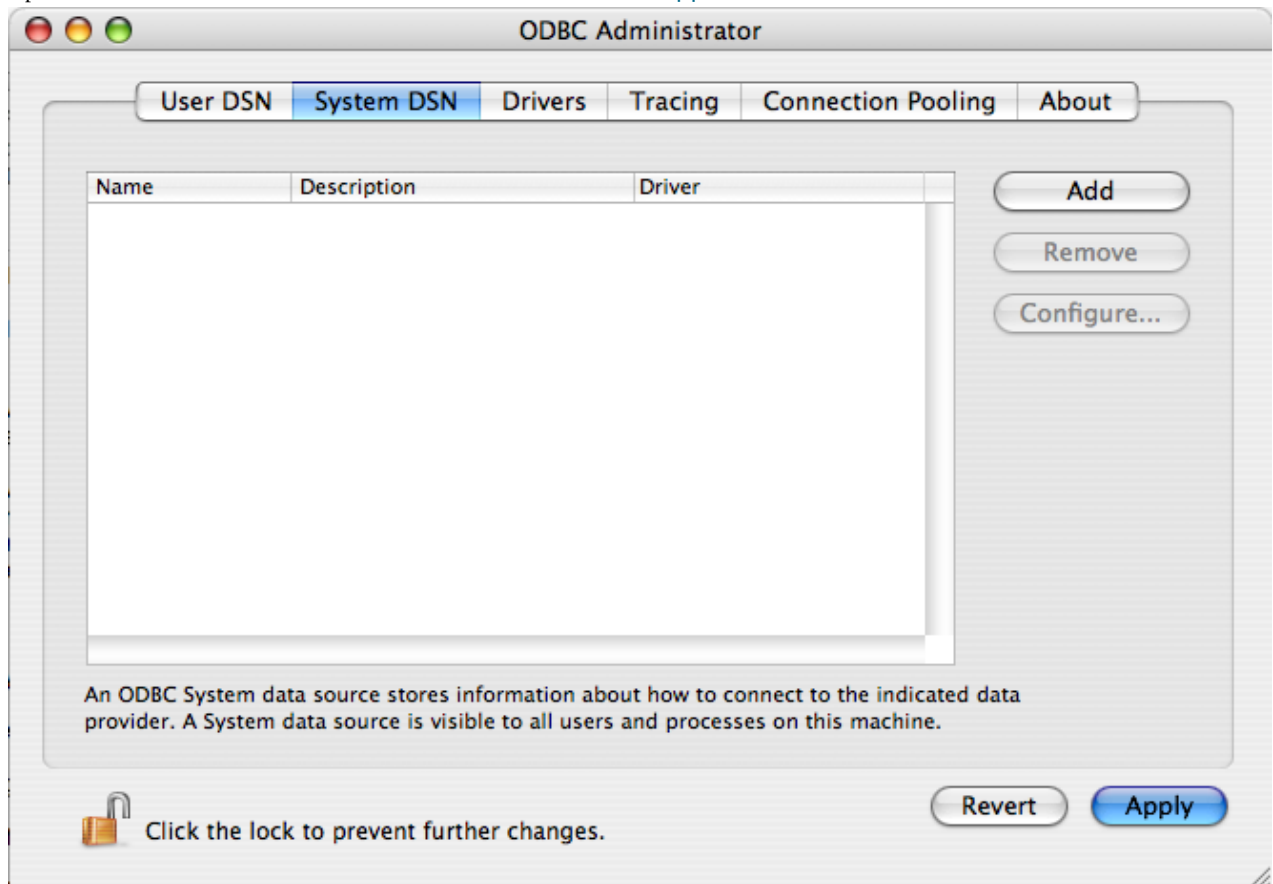
There are known issues with the OS X ODBC Administrator and Connector/ODBC that may prevent you from creating a DSN using this method. In this case you should use the command-line or edit the `odbc.ini` file directly. Note that existing DSNs or those that you create via the `myodbc3i` tool can still be checked and edited using ODBC Administrator.

To create a DSN using the `myodbc3i` utility, you need only specify the DSN type and the DSN connection string. For example:

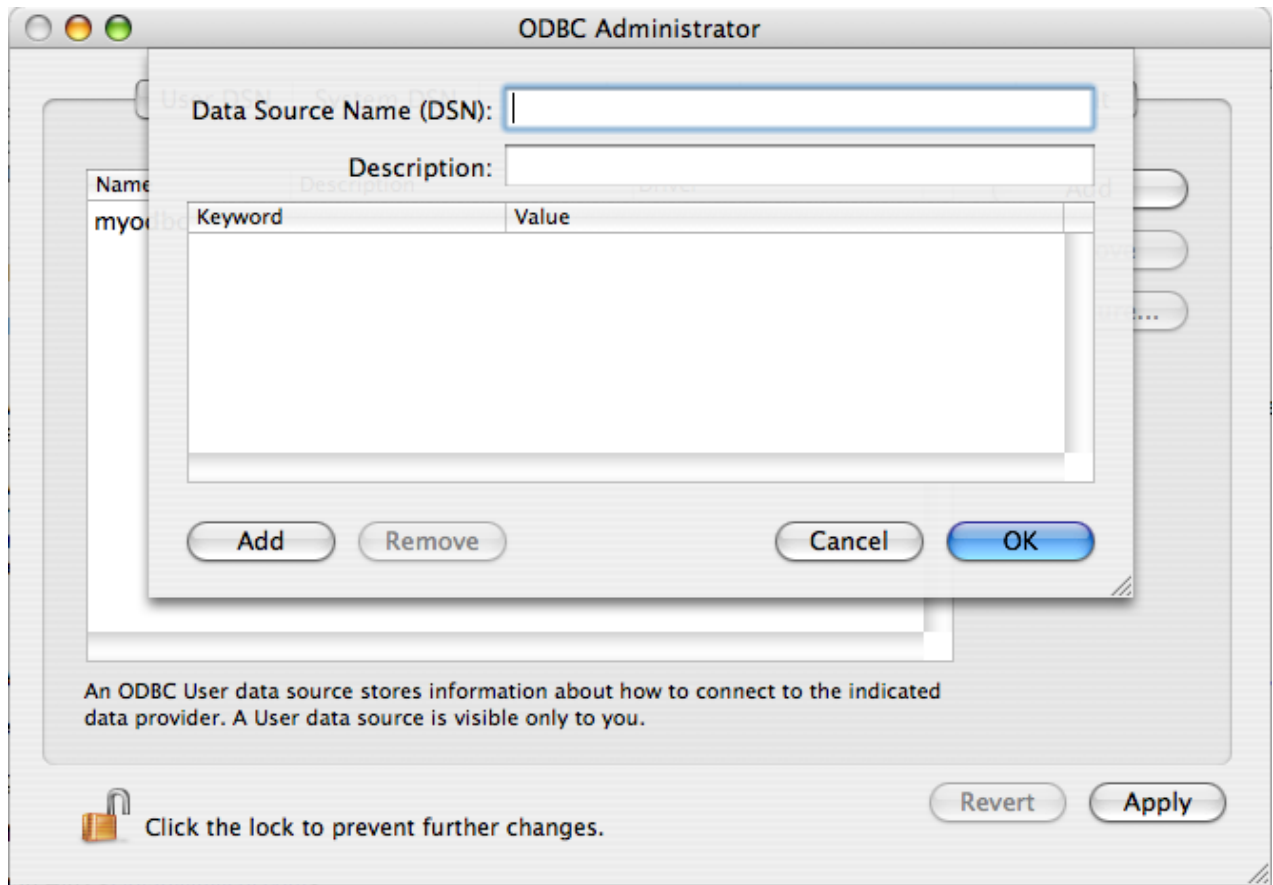
```
$ myodbc3i -a -s -t "DSN=mydb;DRIVER=MySQL ODBC 3.51 Driver;SERVER=mysql;USER=username;PASSWORD=pass"
```

To use ODBC Administrator:

1. Open the ODBC Administrator from the [Utilities](#) folder in the [Applications](#) folder.



2. On the User DSN or System DSN panel, click ADD.
3. Select the Connector/ODBC driver and click OK.
4. You will be presented with the [Data Source Name](#) dialog. Enter The [Data Source Name](#) and an optional [Description](#) for the DSN.



5. Click **ADD** to add a new keyword/value pair to the panel. You should configure at least four pairs to specify the [server](#), [user-name](#), [password](#) and [database](#) connection parameters. See [Section 26.1.3.5, “Connector/ODBC Connection Parameters”](#).
6. Click **OK** to add the DSN to the list of configured data source names.

A completed DSN configuration may look like this:

Data Source Name (DSN): **WorldSample**

Description: **Connection to sample World database**

Keyword	Value
server	mysql
user	sakila
password	Sample
database	test_world

Buttons: Add, Remove, Cancel, OK

You can configure additional ODBC options to your DSN by adding further keyword/value pairs and setting the corresponding values. See [Section 26.1.3.5, “Connector/ODBC Connection Parameters”](#).

26.1.3.4. Configuring a Connector/ODBC DSN on Unix

On **Unix**, you configure DSN entries directly in the `odbc.ini` file. Here is a typical `odbc.ini` file that configures `myodbc` and `myodbc3` as the DSN names for MyODBC 2.50 and Connector/ODBC 3.51, respectively:

```

;
; odbc.ini configuration for Connector/ODBC and Connector/ODBC 3.51 drivers
;
[ODBC Data Sources]
myodbc      = MyODBC 2.50 Driver DSN
myodbc3     = MyODBC 3.51 Driver DSN

[myodbc]
Driver      = /usr/local/lib/libmyodbc.so
Description = MyODBC 2.50 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password    =
Database    = test
OPTION      = 3
SOCKET      =

[myodbc3]
Driver      = /usr/local/lib/libmyodbc3.so
Description = Connector/ODBC 3.51 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password    =
Database    = test
OPTION      = 3
SOCKET      =

[Default]
Driver      = /usr/local/lib/libmyodbc3.so
Description = Connector/ODBC 3.51 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password    =
Database    = test
OPTION      = 3

```

SOCKET =

Refer to the [Section 26.1.3.5, “Connector/ODBC Connection Parameters”](#), for the list of connection parameters that can be supplied.

Note

If you are using [unixODBC](#), you can use the following tools to set up the DSN:

- ODBCConfig GUI tool([HOWTO: ODBCConfig](#))
- `odbcinst`

In some cases when using [unixODBC](#), you might get this error:

```
Data source name not found and no default driver specified
```

If this happens, make sure the `ODBCINI` and `ODBCSYSINI` environment variables are pointing to the right `odbc.ini` file. For example, if your `odbc.ini` file is located in `/usr/local/etc`, set the environment variables like this:

```
export ODBCINI=/usr/local/etc/odbc.ini
export ODBCSYSINI=/usr/local/etc
```

26.1.3.5. Connector/ODBC Connection Parameters

You can specify the parameters in the following tables for Connector/ODBC when configuring a DSN. Users on Windows can use the Options and Advanced panels when configuring a DSN to set these parameters; see the table for information on which options relate to which fields and checkboxes. On Unix and Mac OS X, use the parameter name and value as the keyword/value pair in the DSN configuration. Alternatively, you can set these parameters within the `InConnectionString` argument in the `SQLDriverConnect()` call.

Parameter	Default Value	Comment
<code>user</code>	ODBC (on Windows)	The username used to connect to MySQL.
<code>uid</code>	ODBC (on Windows)	Synonymous with <code>user</code> . Added in 3.51.16.
<code>server</code>	<code>localhost</code>	The hostname of the MySQL server.
<code>database</code>		The default database.
<code>option</code>	0	Options that specify how Connector/ODBC should work. See below.
<code>port</code>	3306	The TCP/IP port to use if <code>server</code> is not <code>localhost</code> .
<code>stmt</code>		A statement to execute when connecting to MySQL.
<code>password</code>		The password for the <code>user</code> account on <code>server</code> .
<code>pwd</code>		Synonymous with <code>password</code> . Added in 3.51.16.
<code>socket</code>		The Unix socket file or Windows named pipe to connect to if <code>server</code> is <code>localhost</code> .
<code>sslca</code>		The path to a file with a list of trust SSL CAs. Added in 3.51.16.
<code>sslcapath</code>		The path to a directory that contains trusted SSL CA certificates in PEM format. Added in 3.51.16.
<code>sslcert</code>		The name of the SSL certificate file to use for establishing a secure connection. Added in 3.51.16.
<code>sslcipher</code>		A list of allowable ciphers to use for SSL encryption. The cipher list has the same format as the <code>openssl ciphers</code> command. Added in 3.51.16.
<code>sslkey</code>		The name of the SSL key file to use for establishing a secure connection. Added in 3.51.16.
<code>charset</code>		The character set to use for the connection. Added in 3.51.17.

Note

The SSL configuration parameters can also be automatically loaded from a `my.ini` or `my.cnf` file.

The `option` argument is used to tell Connector/ODBC that the client isn't 100% ODBC compliant. On Windows, you normally select options by toggling the checkboxes in the connection screen, but you can also select them in the `option` argument. The following options are listed in the order in which they appear in the Connector/ODBC connect screen:

Value	Flagname	GUI Option	Description
1	<code>FLAG_FIELD_LENGTH</code>	Don't Optimize Column Width	The client can't handle that Connector/ODBC returns the real width of a column.
2	<code>FLAG_FOUND_ROWS</code>	Return Matching Rows	The client can't handle that MySQL returns the true value of affected rows. If this flag is set, MySQL returns "found rows" instead. You must have MySQL 3.21.14 or newer to get this to work.
4	<code>FLAG_DEBUG</code>	Trace Driver Calls To myodbc.log	Make a debug log in <code>C:\myodbc.log</code> on Windows, or <code>/tmp/myodbc.log</code> on Unix variants.
8	<code>FLAG_BIG_PACKETS</code>	Allow Big Results	Don't set any packet limit for results and parameters.
16	<code>FLAG_NO_PROMPT</code>	Don't Prompt Upon Connect	Don't prompt for questions even if driver would like to prompt.
32	<code>FLAG_DYNAMIC_CURSOR</code>	Enable Dynamic Cursor	Enable or disable the dynamic cursor support. (Not allowed in Connector/ODBC 2.50.)
64	<code>FLAG_NO_SCHEMA</code>	Ignore # in Table Name	Ignore use of database name in <code>db_name.tbl_name.col_name</code> .
128	<code>FLAG_NO_DEFAULT_CURSOR</code>	User Manager Cursors	Force use of ODBC manager cursors (experimental).
256	<code>FLAG_NO_LOCALE</code>	Don't Use Set Locale	Disable the use of extended fetch (experimental).
512	<code>FLAG_PAD_SPACE</code>	Pad Char To Full Length	Pad <code>CHAR</code> columns to full column length.
1024	<code>FLAG_FULL_COLUMN_NAMES</code>	Return Table Names for SQLDescribeCol	<code>SQLDescribeCol()</code> returns fully qualified column names.
2048	<code>FLAG_COMPRESSED_PROTOCOL</code>	Use Compressed Protocol	Use the compressed client/server protocol.
4096	<code>FLAG_IGNORE_SPACE</code>	Ignore Space After Function Names	Tell server to ignore space after function name and before '(' (needed by PowerBuilder). This makes all function names keywords.
8192	<code>FLAG_NAMED_PIPE</code>	Force Use of Named Pipes	Connect with named pipes to a <code>mysqld</code> server running on NT.
16384	<code>FLAG_NO_BIGINT</code>	Change BIGINT Columns to Int	Change <code>BIGINT</code> columns to <code>INT</code> columns (some applications can't handle <code>BIGINT</code>).
32768	<code>FLAG_NO_CATALOG</code>	No Catalog	Forces results from the catalog functions, such as <code>SQLTables</code> , to always return <code>NULL</code> and the driver to report that catalogs are not supported.
65536	<code>FLAG_USE_MYCNF</code>	Read Options From my.cnf	Read parameters from the <code>[client]</code> and <code>[odbc]</code> groups from <code>my.cnf</code> .
131072	<code>FLAG_SAFE</code>	Safe	Add some extra safety checks.
262144	<code>FLAG_NO_TRANSACTIONS</code>	Disable transactions	Disable transactions.
524288	<code>FLAG_LOG_QUERY</code>	Save queries to myodbc.sql	Enable query logging to <code>c:\myodbc.sql(/tmp/myodbc.sql)</code> file. (Enabled only in debug mode.)
1048576	<code>FLAG_NO_CACHE</code>	Don't Cache Result (forward only cursors)	Do not cache the results locally in the driver, instead read from server (<code>mysql_use_result()</code>). This works only for forward-only cursors. This option is very important in dealing with large tables when you don't want the driver to cache the entire result set.
2097152	<code>FLAG_FORWARD_CURSOR</code>	Force Use Of Forward Only Cursors	Force the use of <code>Forward-only</code> cursor type. In case of applications setting the default static/dynamic cursor type, and one wants

			the driver to use non-cache result sets, then this option ensures the forward-only cursor behavior.
4194304	FLAG_AUTO_RECONNECT	Enable auto-reconnect.	Enables auto-reconnection functionality. You should not use this option with transactions, since a auto reconnection during a incomplete transaction may cause corruption. Note that an auto-reconnected connection will not inherit the same settings and environment as the original. This option was enabled in Connector/ODBC 3.5.13.
8388608	FLAG_AUTO_IS_NULL	Flag Auto Is Null	When set, this option causes the connection to set the SQL_AUTO_IS_NULL option to 1. This disables the standard behavior, but may enable older applications to correctly identify AUTO_INCREMENT values. For more information. See IS NULL . This option was enabled in Connector/ODBC 3.5.13.
16777216	FLAG_ZERO_DATE_TO_MIN	Flag Zero Date to Min	Translates zero dates (XXXX-00-00) into the minimum date values supported by ODBC, XXXX-01-01 . This resolves an issue where some statement swill not work because the date returned and the minimumd ODBC date value are incompatible. This option was enabled in Connector/ODBC 3.5.17.
33554432	FLAG_MIN_DATE_TO_ZERO	Flag Min Date to Zero	Translates the minimum ODBC date value (XXXX-01-01) to the zero date format supported by MySQL (XXXX-00-00). This resolves an issue where some statement swill not work because the date returned and the minimumd ODBC date value are incompatible. This option was enabled in Connector/ODBC 3.5.17.
67108864	FLAG_MULTI_STATEMENTS	Allow multiple statements	Enables support for batched statements. This option was enabled in Connector/ODBC 3.5.18.

To select multiple options, add together their values. For example, setting [option](#) to 12 (4+8) gives you debugging without packet limits.

The following table shows some recommended [option](#) values for various configurations:

Configuration	Option Value
Microsoft Access, Visual Basic	3
Driver trace generation (Debug mode)	4
Microsoft Access (with improved DELETE queries)	35
Large tables with too many rows	2049
Sybase PowerBuilder	135168
Query log generation (Debug mode)	524288
Generate driver trace as well as query log (Debug mode)	524292
Large tables with no-cache results	3145731

26.1.3.6. Connecting Without a Predefined DSN

You can connect to the MySQL server using `SQLDriverConnect`, by specifying the [DRIVER](#) name field. Here are the connection strings for Connector/ODBC using DSN-Less connections:

For MyODBC 2.50:

```
ConnectionString = "DRIVER={MySQL};\
SERVER=localhost;\
DATABASE=test;\
USER=venu;\
PASSWORD=venu;\
OPTION=3;"
```

For Connector/ODBC 3.51:


```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};\n    SERVER=localhost;\n    DATABASE=test;\n    USER=venu;\n    PASSWORD=venu;\n    OPTION=3;"
```

If your programming language converts backslash followed by whitespace to a space, it is preferable to specify the connection string as a single long string, or to use a concatenation of multiple strings that does not add spaces in between. For example:

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"\n    "SERVER=localhost;"\n    "DATABASE=test;"\n    "USER=venu;"\n    "PASSWORD=venu;"\n    "OPTION=3;"
```

Note. Note that on Mac OS X you may need to specify the full path to the Connector/ODBC driver library.

Refer to the [Section 26.1.3.5, “Connector/ODBC Connection Parameters”](#), for the list of connection parameters that can be supplied.

26.1.3.7. ODBC Connection Pooling

Connection pooling enables the ODBC driver to re-use existing connections to a given database from a pool of connections, instead of opening a new connection each time the database is accessed. By enabling connection pooling you can improve the overall performance of your application by lowering the time taken to open a connection to a database in the connection pool.

For more information about connection pooling: <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q169470>.

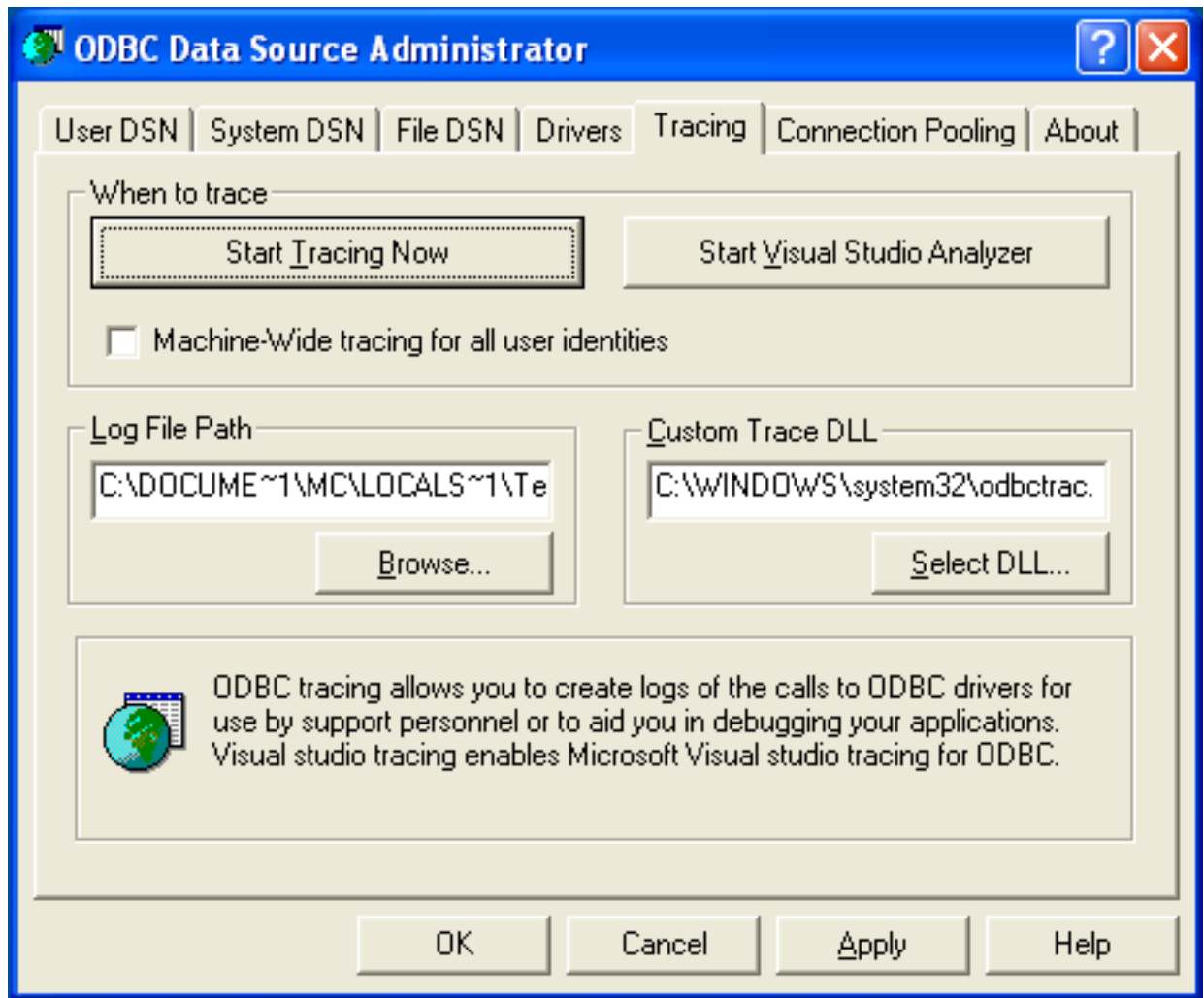
26.1.3.8. Getting an ODBC Trace File

If you encounter difficulties or problems with Connector/ODBC, you should start by making a log file from the [ODBC Manager](#) and Connector/ODBC. This is called *tracing*, and is enabled through the ODBC Manager. The procedure for this differs for Windows, Mac OS X and Unix.

26.1.3.8.1. Enabling ODBC Tracing on Windows

To enable the trace option on Windows:

1. The [Tracing](#) tab of the ODBC Data Source Administrator dialog box enables you to configure the way ODBC function calls are traced.

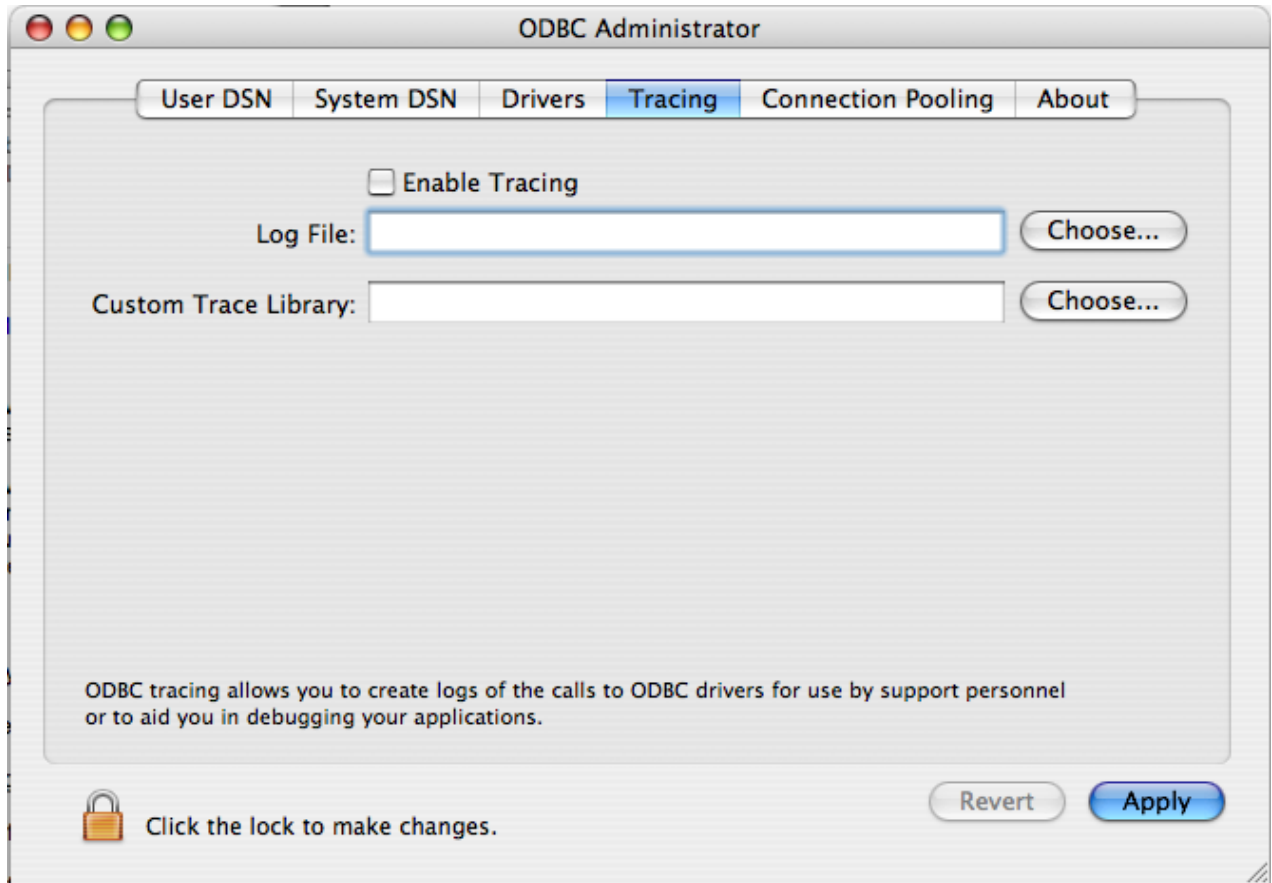


2. When you activate tracing from the **Tracing** tab, the **Driver Manager** logs all ODBC function calls for all subsequently run applications.
3. ODBC function calls from applications running before tracing is activated are not logged. ODBC function calls are recorded in a log file you specify.
4. Tracing ceases only after you click **Stop Tracing Now**. Remember that while tracing is on, the log file continues to increase in size and that tracing affects the performance of all your ODBC applications.

26.1.3.8.2. Enabling ODBC Tracing on Mac OS X

To enable the trace option on Mac OS X 10.3 or later you should use the **Tracing** tab within ODBC Administrator .

1. Open the ODBC Administrator.
2. Select the **Tracing** tab.



3. Select the **Enable Tracing** checkbox.
4. Enter the location where you want to save the Tracing log. If you want to append information to an existing log file, click the **CHOOSE...** button.

26.1.3.8.3. Enabling ODBC Tracing on Unix

To enable the trace option on Mac OS X 10.2 (or earlier) or Unix you must add the **trace** option to the ODBC configuration:

1. On Unix, you need to explicitly set the **Trace** option in the **ODBC.INI** file.

Set the tracing **ON** or **OFF** by using **TraceFile** and **Trace** parameters in **odbc.ini** as shown below:

```
TraceFile = /tmp/odbc.trace
Trace     = 1
```

TraceFile specifies the name and full path of the trace file and **Trace** is set to **ON** or **OFF**. You can also use **1** or **YES** for **ON** and **0** or **NO** for **OFF**. If you are using **ODBCConfig** from **unixODBC**, then follow the instructions for tracing **unixODBC** calls at [HOWTO-ODBCConfig](#).

26.1.3.8.4. Enabling a Connector/ODBC Log

To generate a Connector/ODBC log, do the following:

1. Within Windows, enable the **Trace Connector/ODBC** option flag in the Connector/ODBC connect/configure screen. The log is written to file **C:\myodbc.log**. If the trace option is not remembered when you are going back to the above screen, it means

that you are not using the `myodbc.dll` driver, see [Section 26.1.3.2.4, “Errors and Debugging”](#).

On Mac OS X, Unix, or if you are using DSN-Less connection, then you need to supply `OPTION=4` in the connection string or set the corresponding keyword/value pair in the DSN.

2. Start your application and try to get it to fail. Then check the Connector/ODBC trace file to find out what could be wrong.

If you need help determining what is wrong, see [Section 26.1.7.1, “Connector/ODBC Community Support”](#).

26.1.4. Connector/ODBC Examples

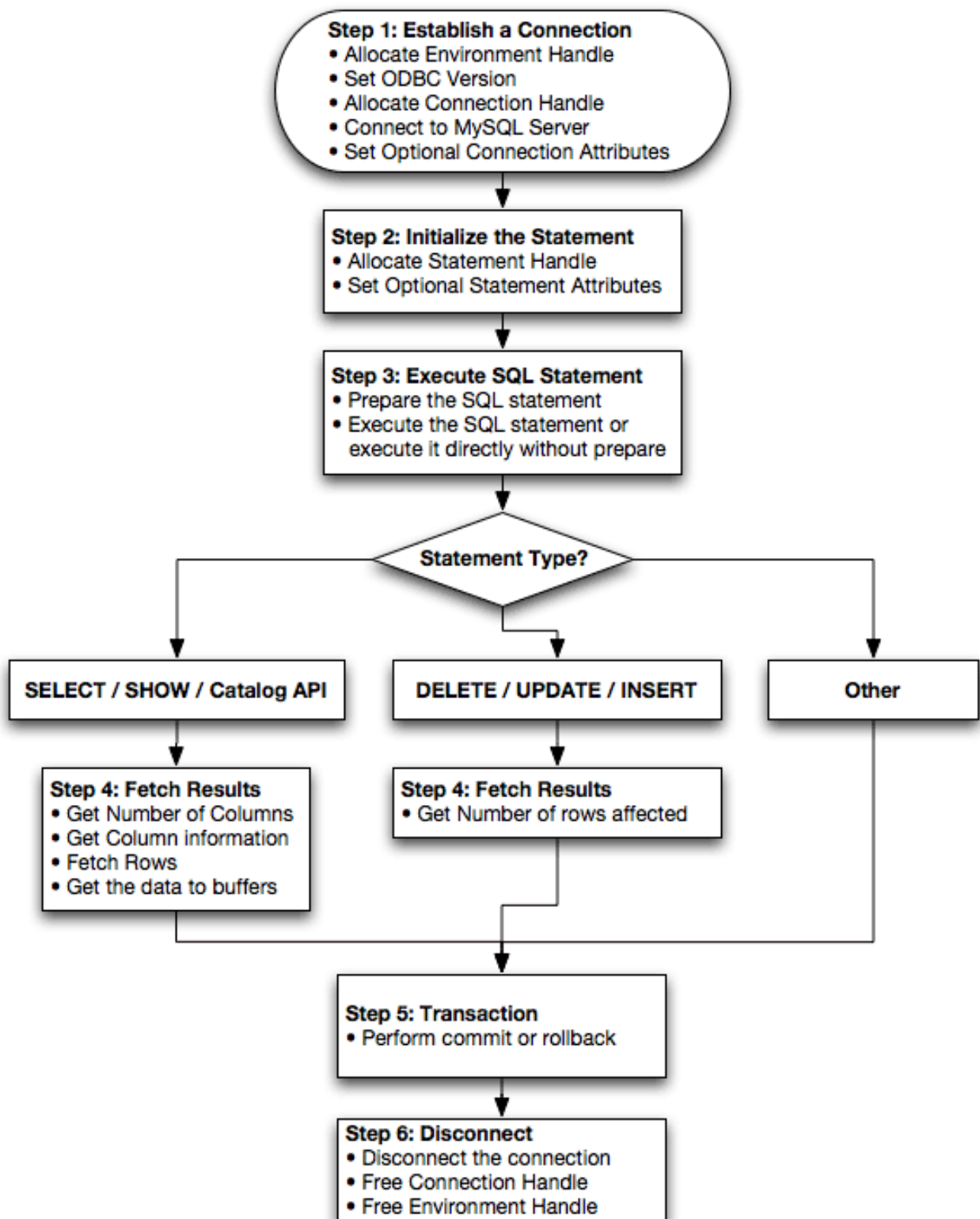
Once you have configured a DSN to provide access to a database, how you access and use that connection is dependent on the application or programming language. As ODBC is a standardized interface, any application or language that supports ODBC can use the DSN and connect to the configured database.

26.1.4.1. Basic Connector/ODBC Application Steps

Interacting with a MySQL server from an applications using the Connector/ODBC typically involves the following operations:

- Configure the Connector/ODBC DSN
- Connect to MySQL server
- Initialization operations
- Execute SQL statements
- Retrieve results
- Perform Transactions
- Disconnect from the server

Most applications use some variation of these steps. The basic application steps are shown in the following diagram:



26.1.4.2. Step-by-step Guide to Connecting to a MySQL Database through Connector/ODBC

A typical installation situation where you would install Connector/ODBC is when you want to access a database on a Linux or Unix host from a Windows machine.

As an example of the process required to set up access between two machines, the steps below take you through the basic steps. These instructions assume that you want to connect to system ALPHA from system BETA with a username and password of **myuser** and **mypassword**.

On system ALPHA (the MySQL server) follow these steps:

1. Start the MySQL server.
2. Use **GRANT** to set up an account with a username of **myuser** that can connect from system BETA using a password of **myuser** to the database **test**:

```
GRANT ALL ON test.* to 'myuser'@'BETA' IDENTIFIED BY 'mypassword';
```

For more information about MySQL privileges, refer to [Section 5.8, “MySQL User Account Management”](#).

On system BETA (the Connector/ODBC client), follow these steps:

1. Configure a Connector/ODBC DSN using parameters that match the server, database and authentication information that you have just configured on system ALPHA.

Parameter	Value	Comment
DSN	remote_test	A name to identify the connection.
SERVER	ALPHA	The address of the remote server.
DATABASE	test	The name of the default database.
USER	myuser	The username configured for access to this database.
PASSWORD	mypassword	The password for myuser .

2. Using an ODBC-capable application, such as Microsoft Office, connect to the MySQL server using the DSN you have just created. If the connection fails, use tracing to examine the connection process. See [Section 26.1.3.8, “Getting an ODBC Trace File”](#), for more information.

26.1.4.3. Connector/ODBC and Third-Party ODBC Tools

Once you have configured your Connector/ODBC DSN, you can access your MySQL database through any application that supports the ODBC interface, including programming languages and third-party applications. This section contains guides and help on using Connector/ODBC with various ODBC-compatible tools and applications, including Microsoft Word, Microsoft Excel and Adobe/Macromedia ColdFusion.

Connector/ODBC has been tested with the following applications:

Publisher	Application	Notes
Adobe	ColdFusion	Formerly Macromedia ColdFusion
Borland	C++ Builder	
	Builder 4	
	Delphi	
Business Objects	Crystal Reports	
Claris	Filemaker Pro	
Corel	Paradox	
Computer Associates	Visual Objects	Also known as CAVO
	AllFusion ERwin Data Modeler	

Gupta	Team Developer	Previously known as Centura Team Developer; Gupta SQL/Windows
Gensym	G2-ODBC Bridge	
Inline	iHTML	
Lotus	Notes	Versions 4.5 and 4.6
Microsoft	Access	
	Excel	
	Visio Enterprise	
	Visual C++	
	Visual Basic	
	ODBC.NET	Using C#, Visual Basic, C++
	FoxPro	
	Visual Interdev	
OpenOffice.org	OpenOffice.org	
Perl	DBD::ODBC	
Pervasive Software	DataJunction	
Sambar Technologies	Sambar Server	
SPSS	SPSS	
SoftVelocity	Clarion	
SQLExpress	SQLExpress for Xbase++	
Sun	StarOffice	
SunSystems	Vision	
Sybase	PowerBuilder	
	PowerDesigner	
theKompany.com	Data Architect	

If you know of any other applications that work with Connector/ODBC, please send mail to <myodbc@lists.mysql.com> about them.

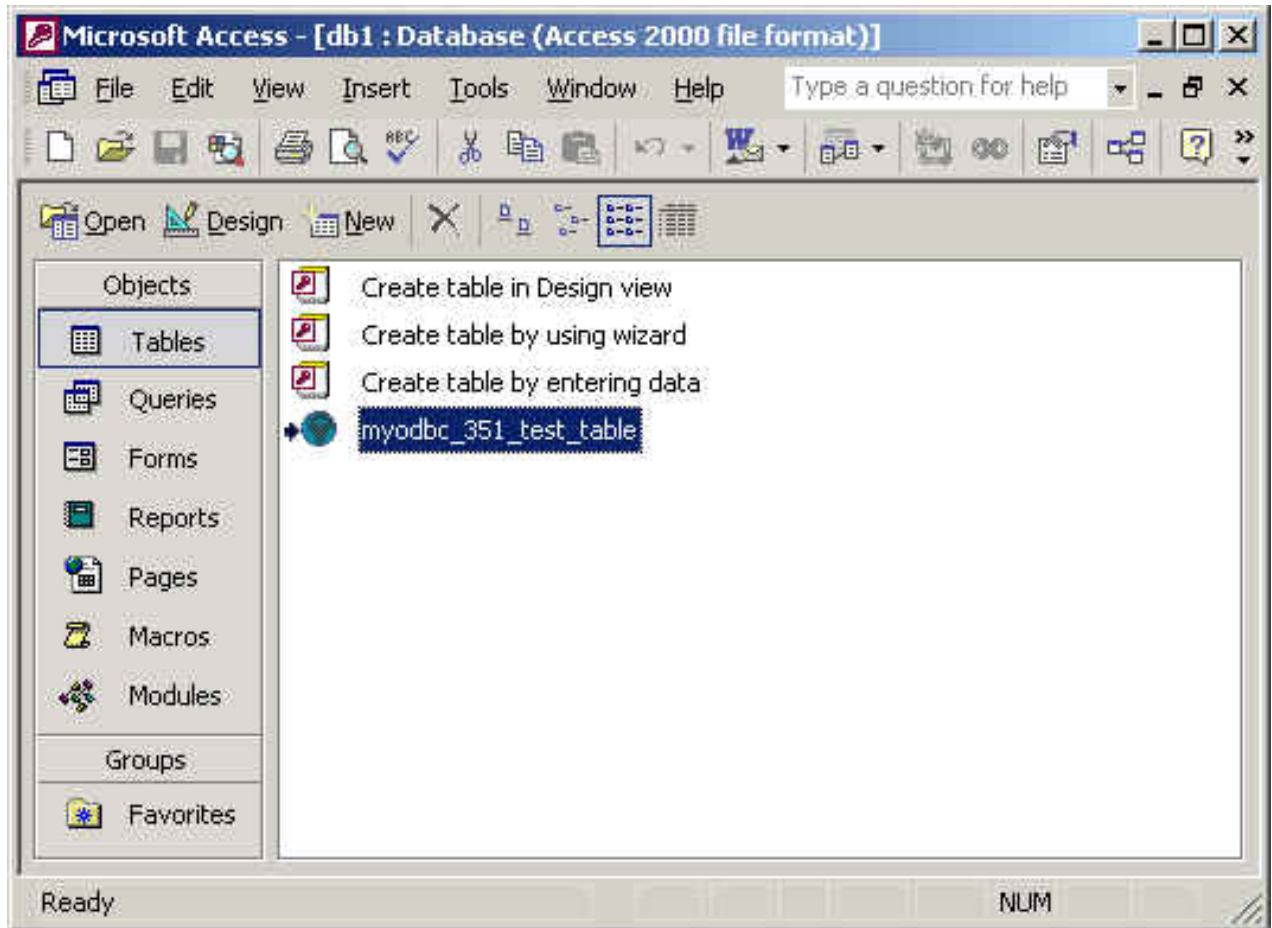
26.1.4.4. Using Connector/ODBC with Microsoft Access

You can use MySQL database with Microsoft Access using Connector/ODBC. The MySQL database can be used as an import source, an export source, or as a linked table for direct use within an Access application, so you can use Access as the front-end interface to a MySQL database.

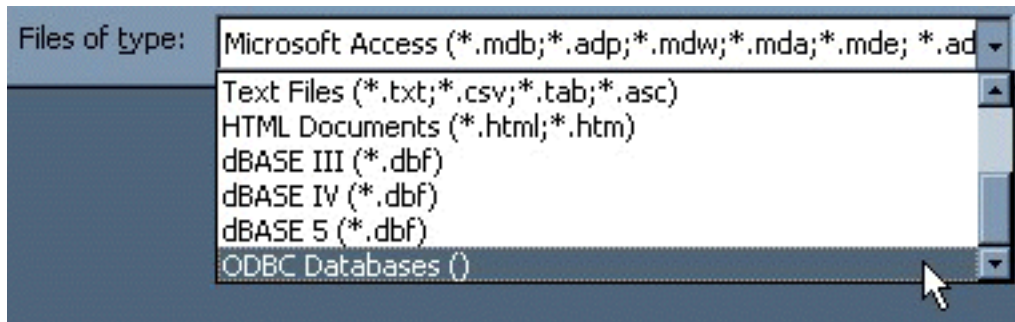
26.1.4.4.1. Exporting Access Data to MySQL

To export a table of data from an Access database to MySQL, follow these instructions:

1. When you open an Access database or an Access project, a Database window appears. It displays shortcuts for creating new database objects and opening existing objects.



2. Click the name of the [table](#) or [query](#) you want to export, and then in the **File** menu, select **Export**.
3. In the **Export Object Type Object name To** dialog box, in the **Save As Type** box, select **ODBC Databases ()** as shown here:



4. In the **Export** dialog box, enter a name for the file (or use the suggested name), and then select **OK**.
5. The **Select Data Source** dialog box is displayed; it lists the defined data sources for any ODBC drivers installed on your computer. Click either the **File Data Source** or **Machine Data Source** tab, and then double-click the **Connector/ODBC** or **Connector/ODBC 3.51** data source that you want to export to. To define a new data source for **Connector/ODBC**, please [Section 26.1.3.2, "Configuring a Connector/ODBC DSN on Windows"](#).

Microsoft Access connects to the MySQL Server through this data source and exports new tables and or data.

26.1.4.4.2. Importing MySQL Data to Access

To import a table or tables from MySQL to Access, follow these instructions:

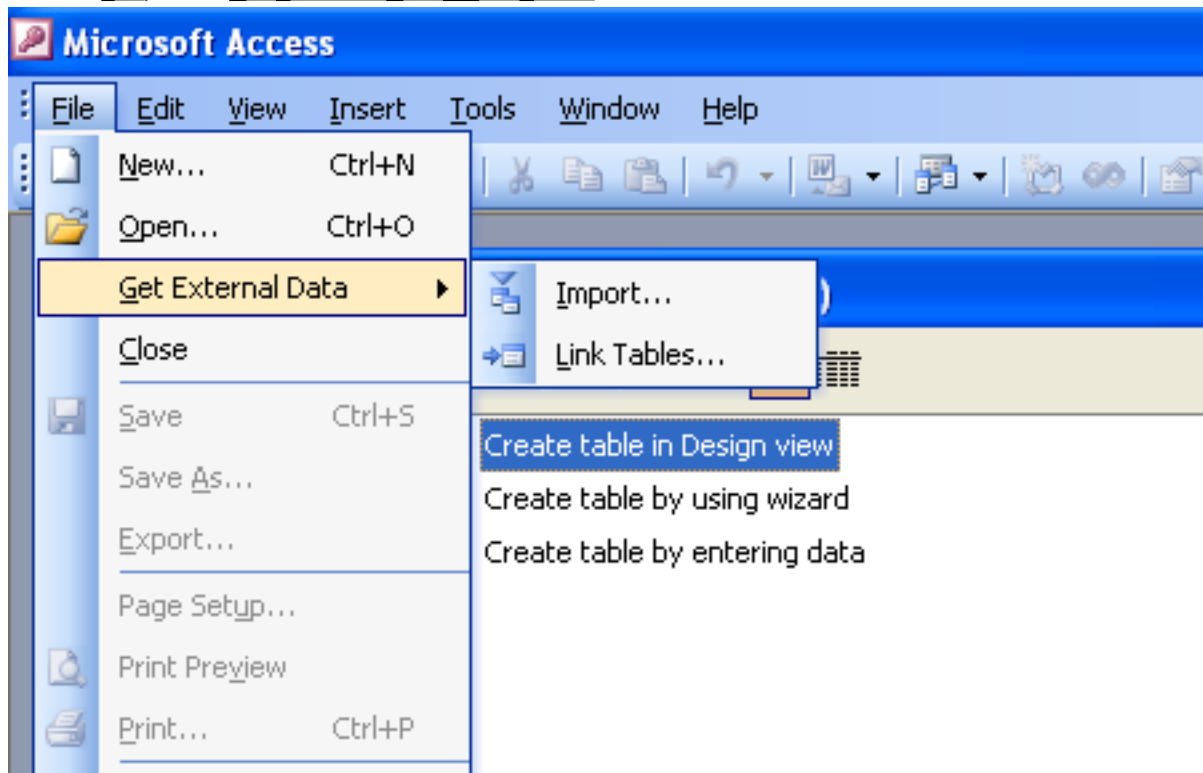
1. Open a database, or switch to the Database window for the open database.
2. To import tables, on the **File** menu, point to **Get External Data**, and then click **Import**.
3. In the **Import** dialog box, in the Files Of Type box, select **ODBC DATABASES ()**. The Select Data Source dialog box lists the defined data sources **THE SELECT DATA SOURCE** dialog box is displayed; it lists the defined data source names.
4. If the ODBC data source that you selected requires you to log on, enter your login ID and password (additional information might also be required), and then click **OK**.
5. Microsoft Access connects to the MySQL server through **ODBC data source** and displays the list of tables that you can **import**.
6. Click each table that you want to **import**, and then click **OK**.

26.1.4.4.3. Using Microsoft Access as a Front-end to MySQL

You can use Microsoft Access as a front end to a MySQL database by linking tables within your Microsoft Access database to tables that exist within your MySQL database. When a query is requested on a table within Access, ODBC is used to execute the queries on the MySQL database instead.

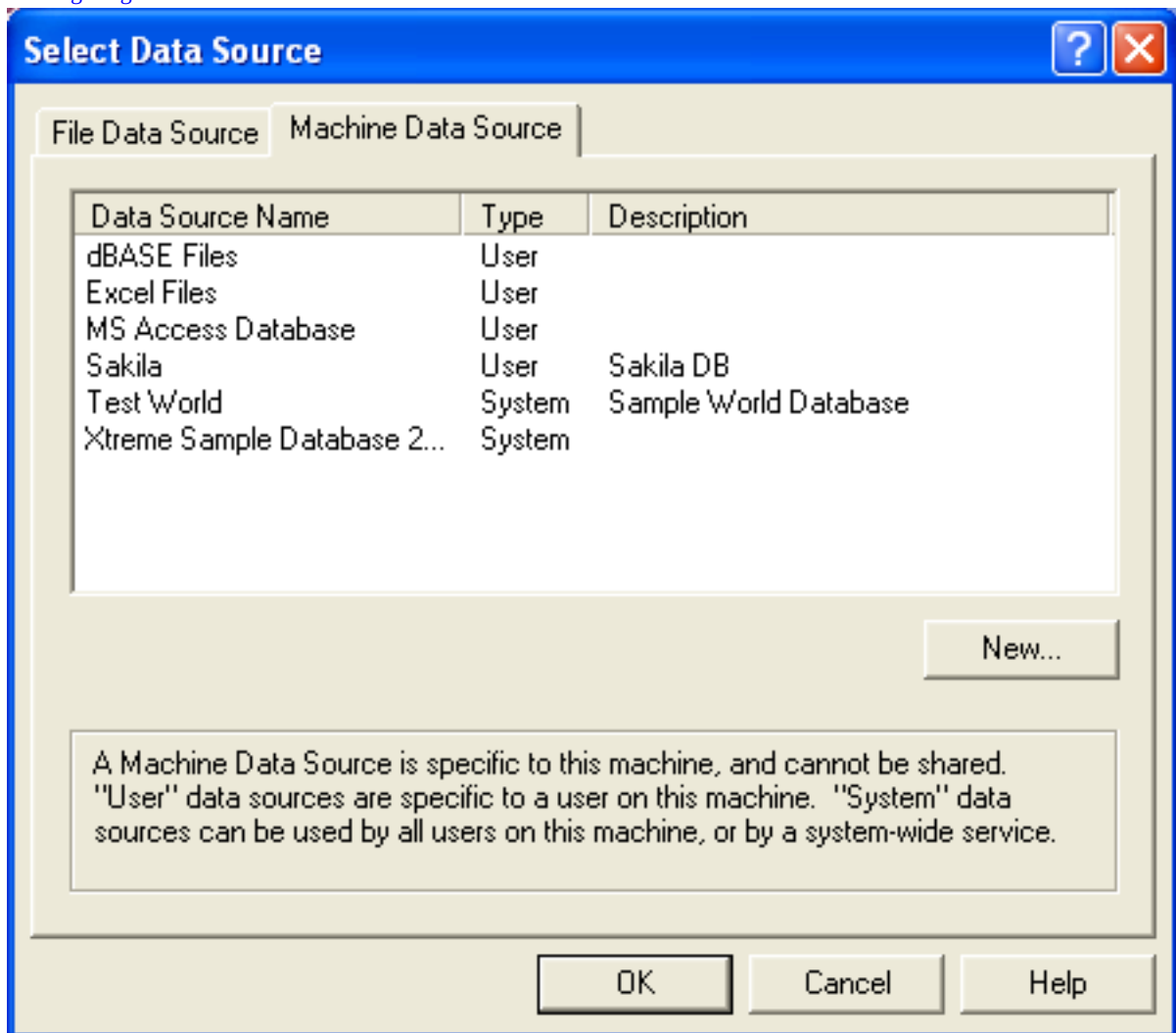
To create a linked table:

1. Open the Access database that you want to link to MySQL.
2. From the **FILE**, choose **GET EXTERNAL DATA->LINK TABLES**.

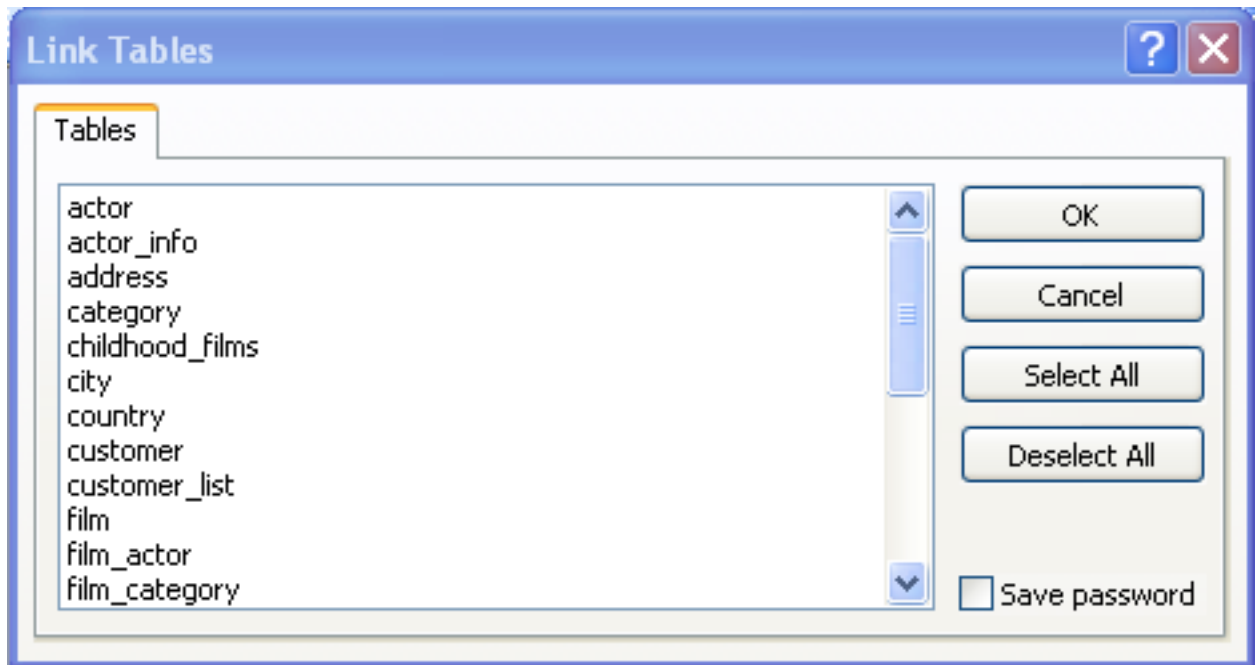


3. From the browser, choose **ODBC DATABASES ()** from the **FILES OF TYPE** popup.
4. In the **SELECT DATA SOURCE** window, choose an existing DSN, either from a **FILE DATA SOURCE** or **MACHINE DATA SOURCE**. You can also create a new DSN using the **NEW...** button. For more information on creating a DSN see [Section 26.1.3.2](#),

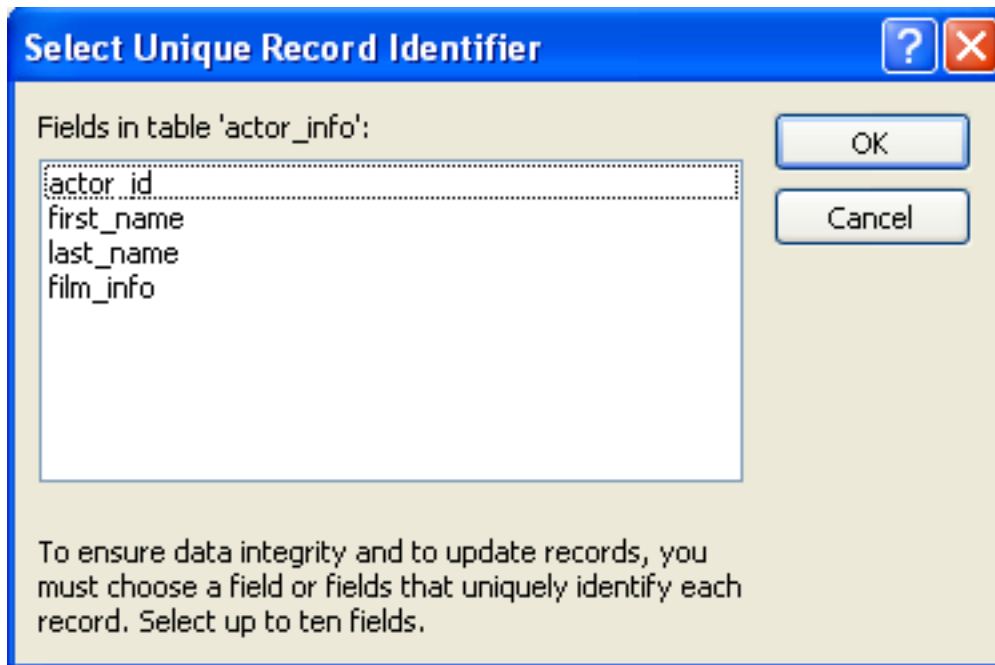
“Configuring a Connector/ODBC DSN on Windows”.



5. In the **LINK TABLES** dialog, select one or more tables from the MySQL database. A link will be created to each table that you select from this list.



6. If Microsoft Access is unable to determine the unique record identifier for a table automatically then it may ask you to confirm the column, or combination of columns, to be used to uniquely identify each row from the source table. Select the columns you want to use and click OK.



Once the process has been completed, you can now build interfaces and queries to the linked tables just as you would for any Access database.

Use the following procedure to view or to refresh links when the structure or location of a linked table has changed. The Linked Table Manager lists the paths to all currently linked tables.

To view or refresh links:

1. Open the database that contains links to MySQL tables.
2. On the **Tools** menu, point to **Add-ins (Database Utilities)** in Access 2000 or newer), and then click **Linked Table Manager**.
3. Select the check box for the tables whose links you want to refresh.
4. Click OK to refresh the links.

Microsoft Access confirms a successful refresh or, if the table wasn't found, displays the **Select New Location of <table name>** dialog box in which you can specify its the table's new location. If several selected tables have moved to the new location that you specify, the Linked Table Manager searches that location for all selected tables, and updates all links in one step.

To change the path for a set of linked tables:

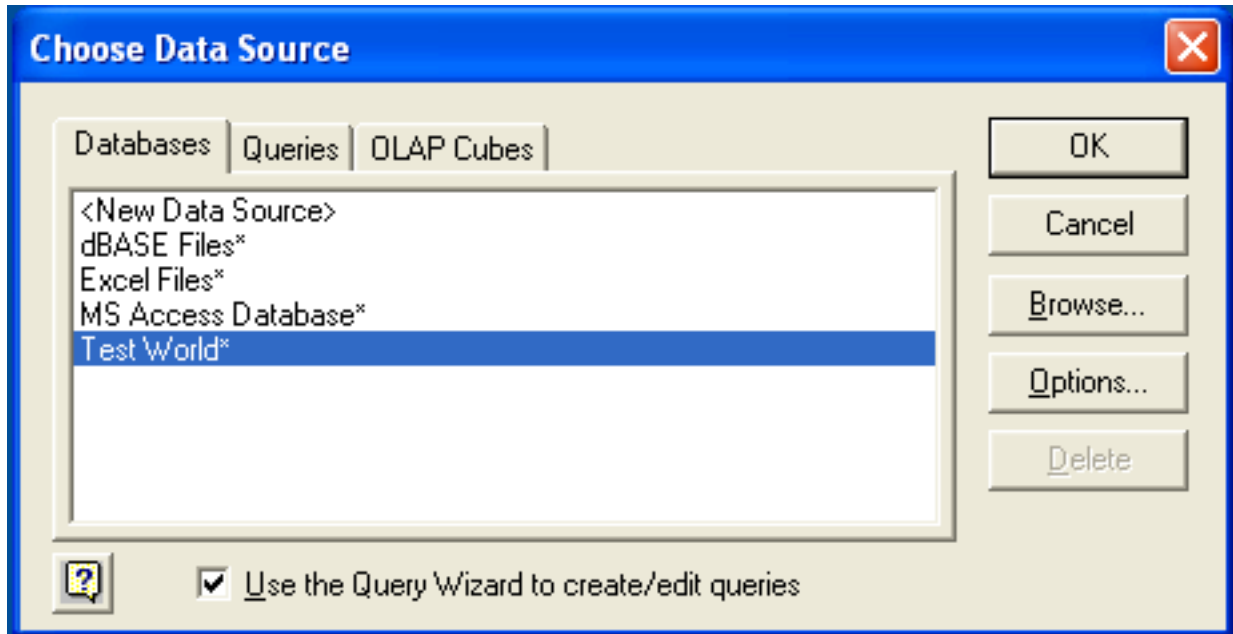
1. Open the database that contains links to tables.
2. On the **Tools** menu, point to **Add-ins (Database Utilities)** in Access 2000 or newer), and then click **Linked Table Manager**.
3. Select the **Always Prompt For A New Location** check box.
4. Select the check box for the tables whose links you want to change, and then click **OK**.
5. In the **Select New Location of <table name>** dialog box, specify the new location, click **Open**, and then click **OK**.

26.1.4.5. Using Connector/ODBC with Microsoft Word or Excel

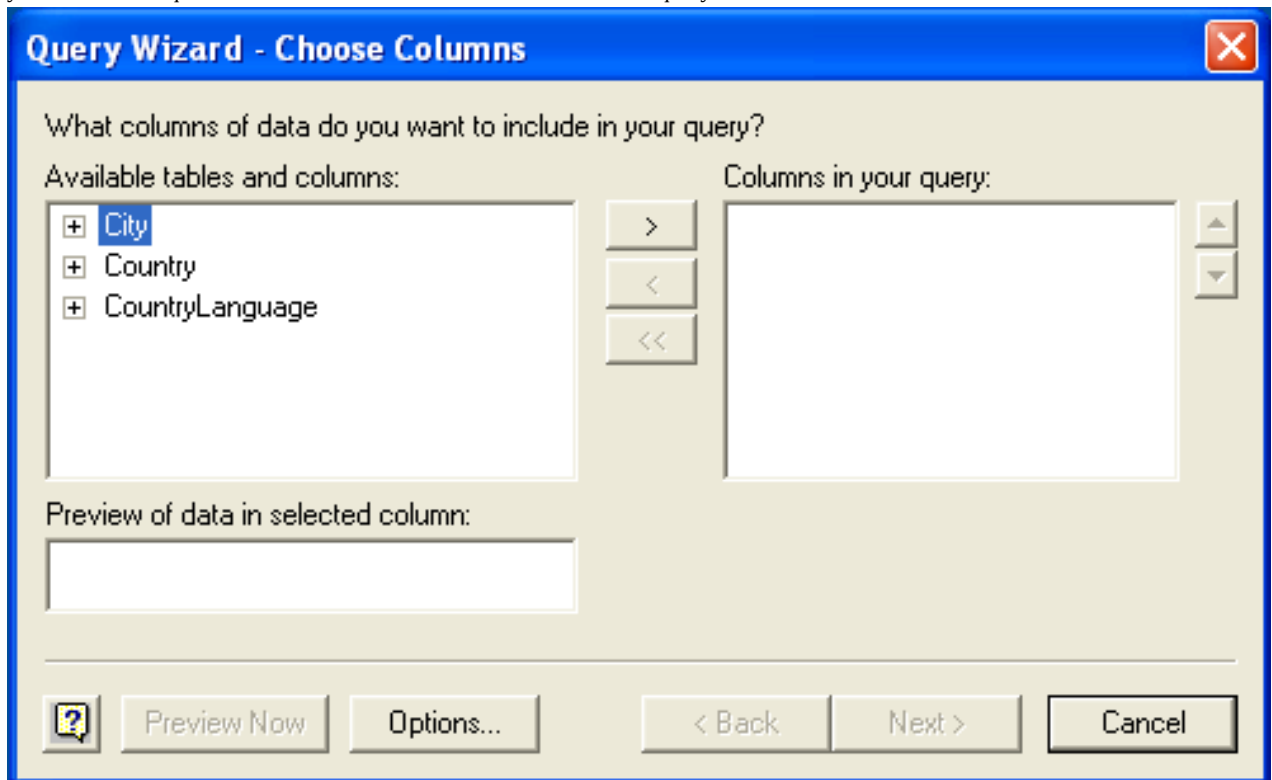
You can use Microsoft Word and Microsoft Excel to access information from a MySQL database using Connector/ODBC. Within Microsoft Word, this facility is most useful when importing data for mailmerge, or for tables and data to be included in reports. Within Microsoft Excel, you can execute queries on your MySQL server and import the data directly into an Excel Worksheet, presenting the data as a series of rows and columns.

With both applications, data is accessed and imported into the application using Microsoft Query , which enables you to execute a query through an ODBC source. You use Microsoft Query to build the SQL statement to be executed, selecting the tables, fields, selection criteria and sort order. For example, to insert information from a table in the World test database into an Excel spreadsheet, using the DSN samples shown in [Section 26.1.3, “Connector/ODBC Configuration”](#):

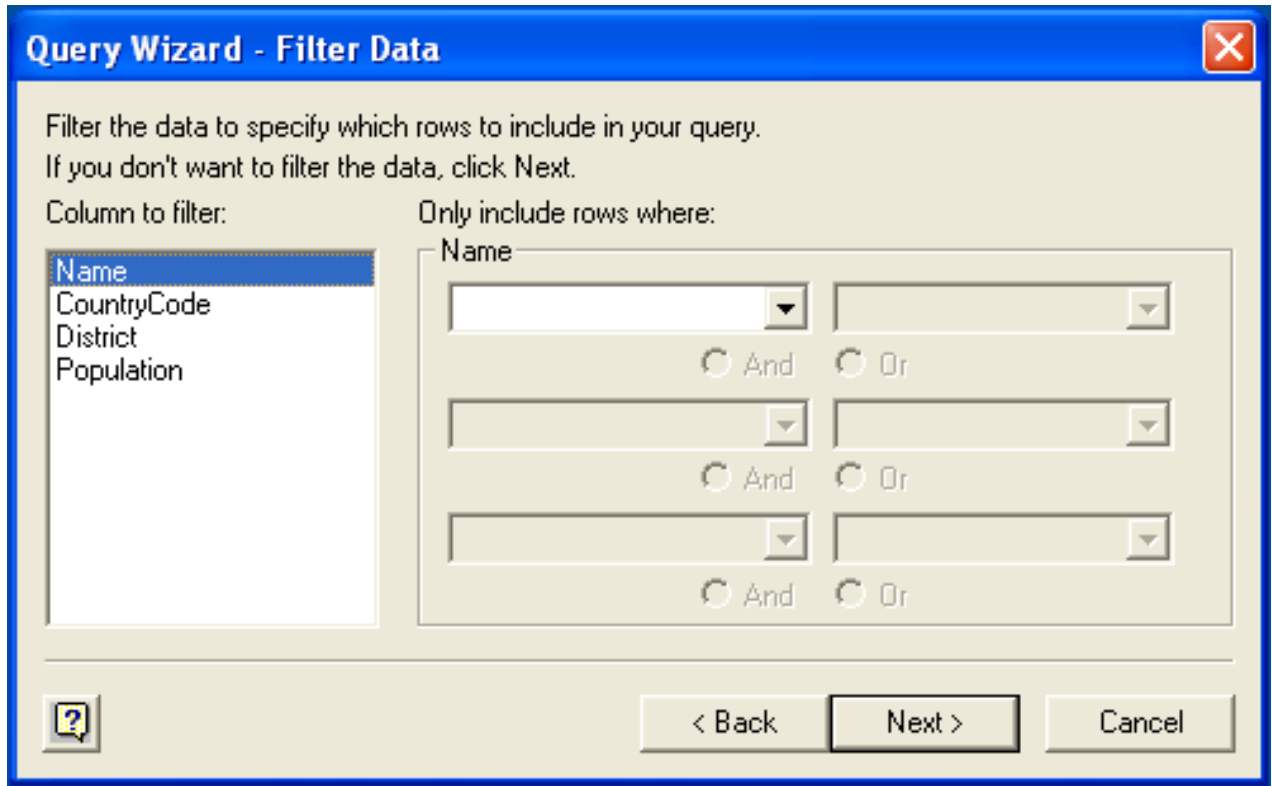
1. Create a new Worksheet.
2. From the **Data** menu, choose **Import External Data**, and then select **New Database Query**.
3. Microsoft Query will start. First, you need to choose the data source, by selecting an existing Data Source Name.



4. Within the [Query Wizard](#), you must choose the columns that you want to import. The list of tables available to the user configured through the DSN is shown on the left, the columns that will be added to your query are shown on the right. The columns you choose are equivalent to those in the first section of a [SELECT](#) query. Click NEXT to continue.



5. You can filter rows from the query (the equivalent of a [WHERE](#) clause) using the [Filter Data](#) dialog. Click NEXT to continue.



Query Wizard - Filter Data

Filter the data to specify which rows to include in your query.
If you don't want to filter the data, click Next.

Column to filter: **Name**
CountryCode
District
Population

Only include rows where:

Name

[Dropdown] [Dropdown]

☐ And ☐ Or

[Dropdown] [Dropdown]

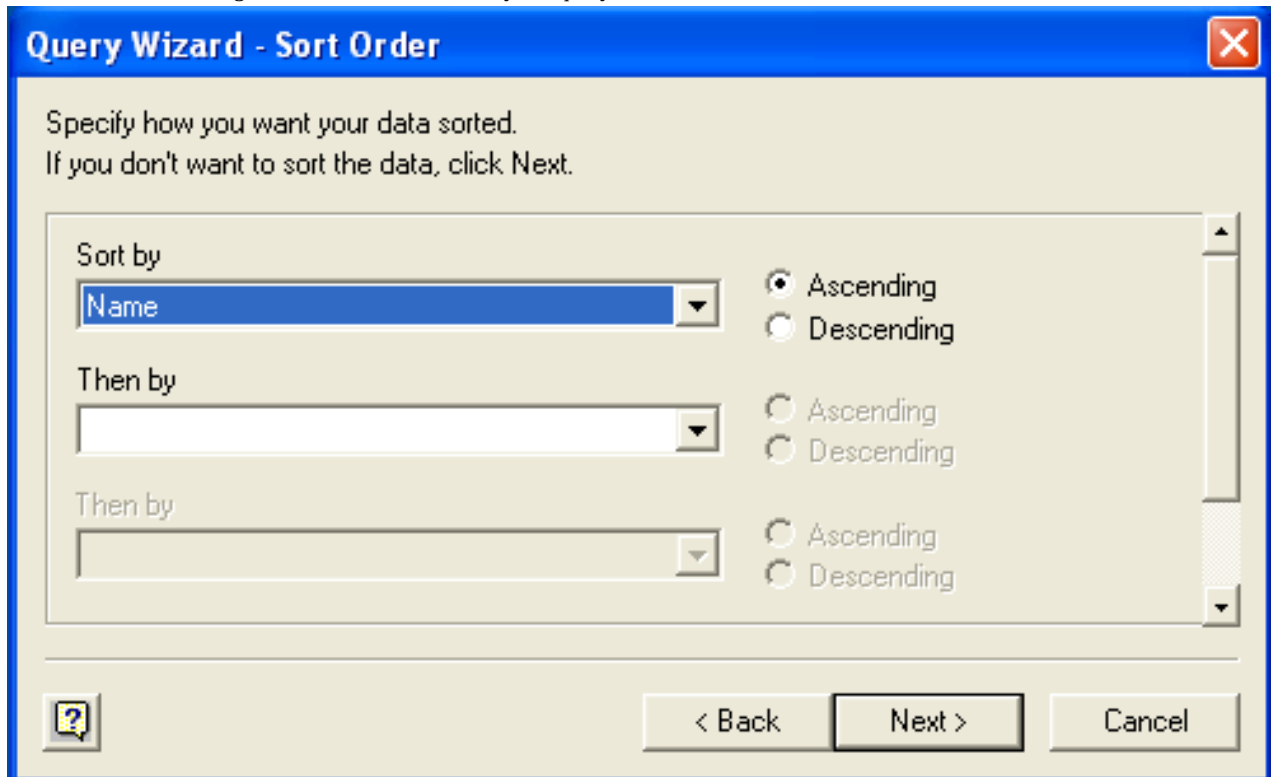
☐ And ☐ Or

[Dropdown] [Dropdown]

☐ And ☐ Or

[?] < Back Next > Cancel

6. Select an (optional) sort order for the data. This is equivalent to using a **ORDER BY** clause in your SQL query. You can select up to three fields for sorting the information returned by the query. Click NEXT to continue.



Query Wizard - Sort Order

Specify how you want your data sorted.
If you don't want to sort the data, click Next.

Sort by

Name [Dropdown]

☒ Ascending
☐ Descending

Then by

[Dropdown]

☐ Ascending
☐ Descending

Then by

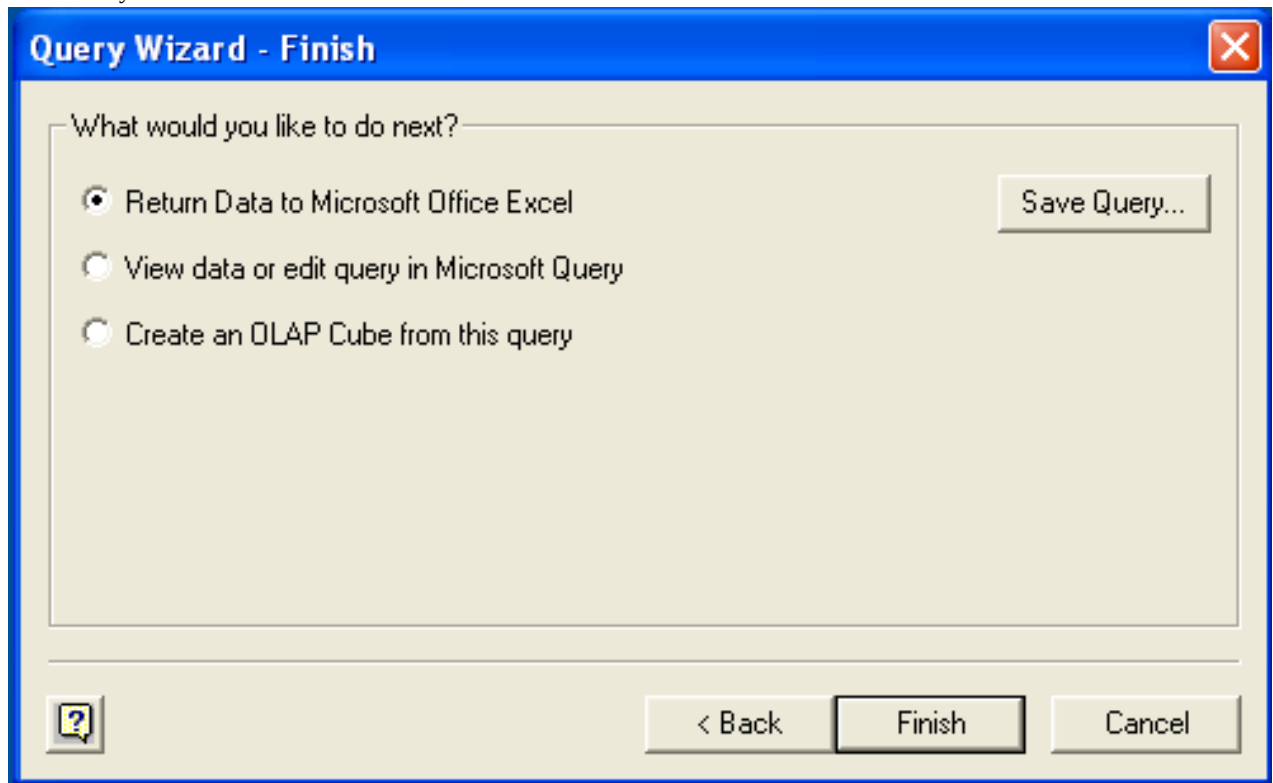
[Dropdown]

☐ Ascending
☐ Descending

[?] < Back Next > Cancel

7. Select the destination for your query. You can select to return the data Microsoft Excel, where you can choose a worksheet and cell

where the data will be inserted; you can continue to view the query and results within Microsoft Query, where you can edit the SQL query and further filter and sort the information returned; or you can create an OLAP Cube from the query, which can then be used directly within Microsoft Excel. Click **FINISH**.



The same process can be used to import data into a Word document, where the data will be inserted as a table. This can be used for mail merge purposes (where the field data is read from a Word table), or where you want to include data and reports within a report or other document.

26.1.4.6. Using Connector/ODBC with Crystal Reports

Crystal Reports can use an ODBC DSN to connect to a database from which you to extract data and information for reporting purposes.

Note

There is a known issue with certain versions of Crystal Reports where the application is unable to open and browse tables and fields through an ODBC connection. Before using Crystal Reports with MySQL, please ensure that you have update to the latest version, including any outstanding service packs and hotfixes. For more information on this issue, see the [Business Objects Knowledgebase](#) for more information.

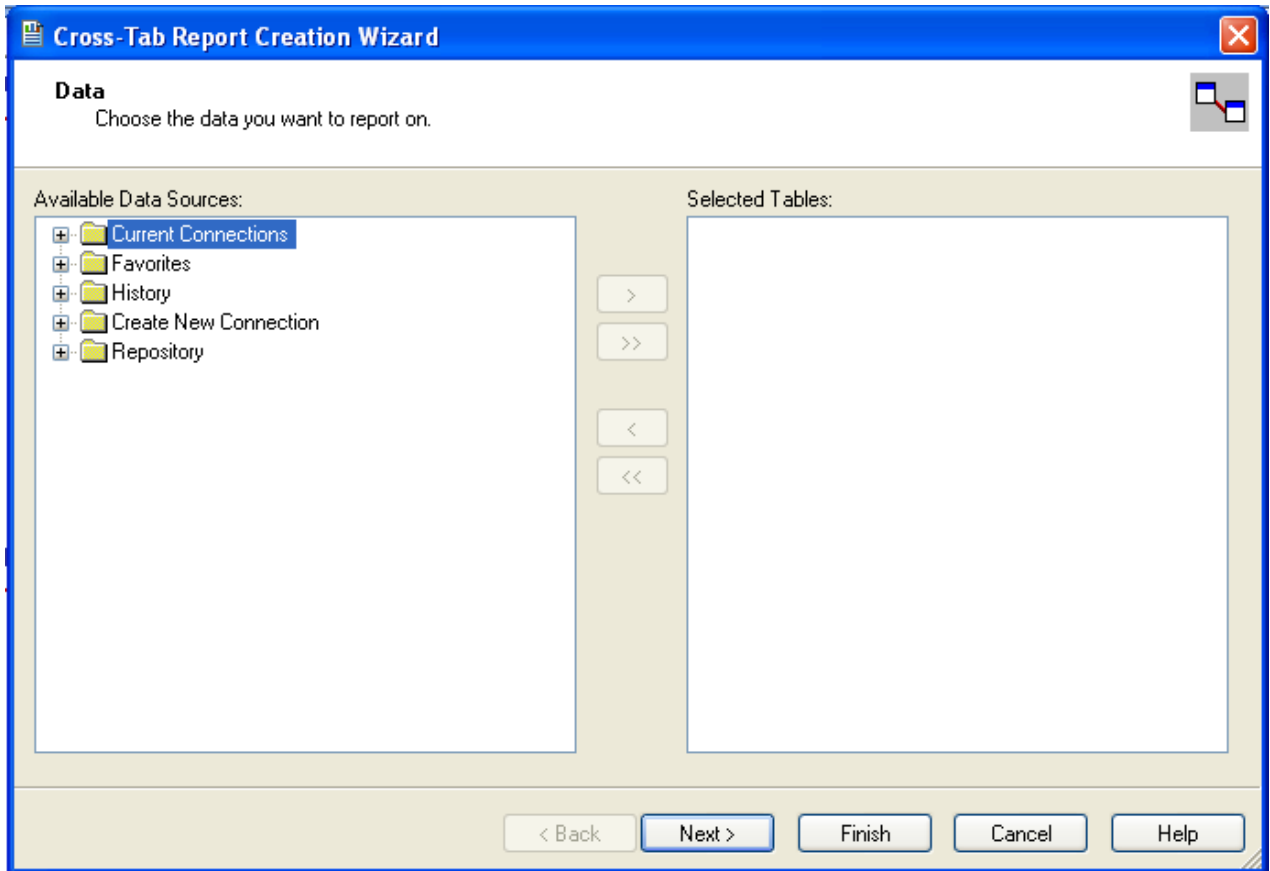
For example, to create a simple crosstab report within Crystal Reports XI, you should follow these steps:

1. Create a DSN using the [Data Sources \(ODBC\)](#) tool. You can either specify a complete database, including username and password, or you can build a basic DSN and use Crystal Reports to set the username and password.

For the purposes of this example, a DSN that provides a connection to an instance of the MySQL Sakila sample database has been created.

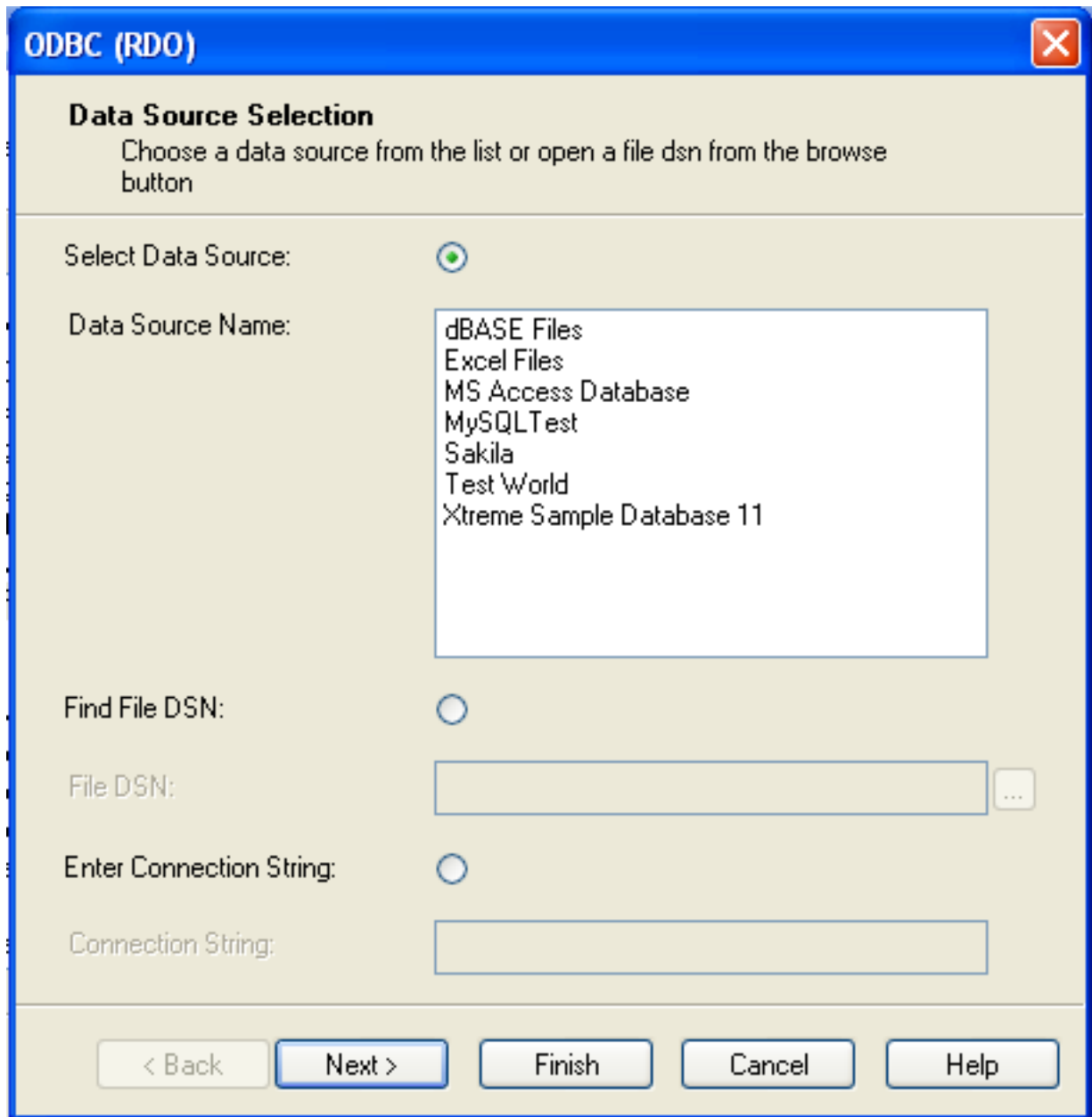
2. Open Crystal Reports and create a new project, or open an existing reporting project into which you want to insert data from your MySQL data source.
3. Start the Cross-Tab Report Wizard, either by clicking on the option on the Start Page. Expand the **CREATE NEW CONNECTION** folder, then expand the **ODBC (RDO)** folder to obtain a list of ODBC data sources.

You will be asked to select a data source.



4. When you first expand the **ODBC (RDO)** folder you will be presented the Data Source Selection screen. From here you can select either a pre-configured DSN, open a file-based DSN or enter and manual connection string. For this example, the **SAKILA** DSN will be used.

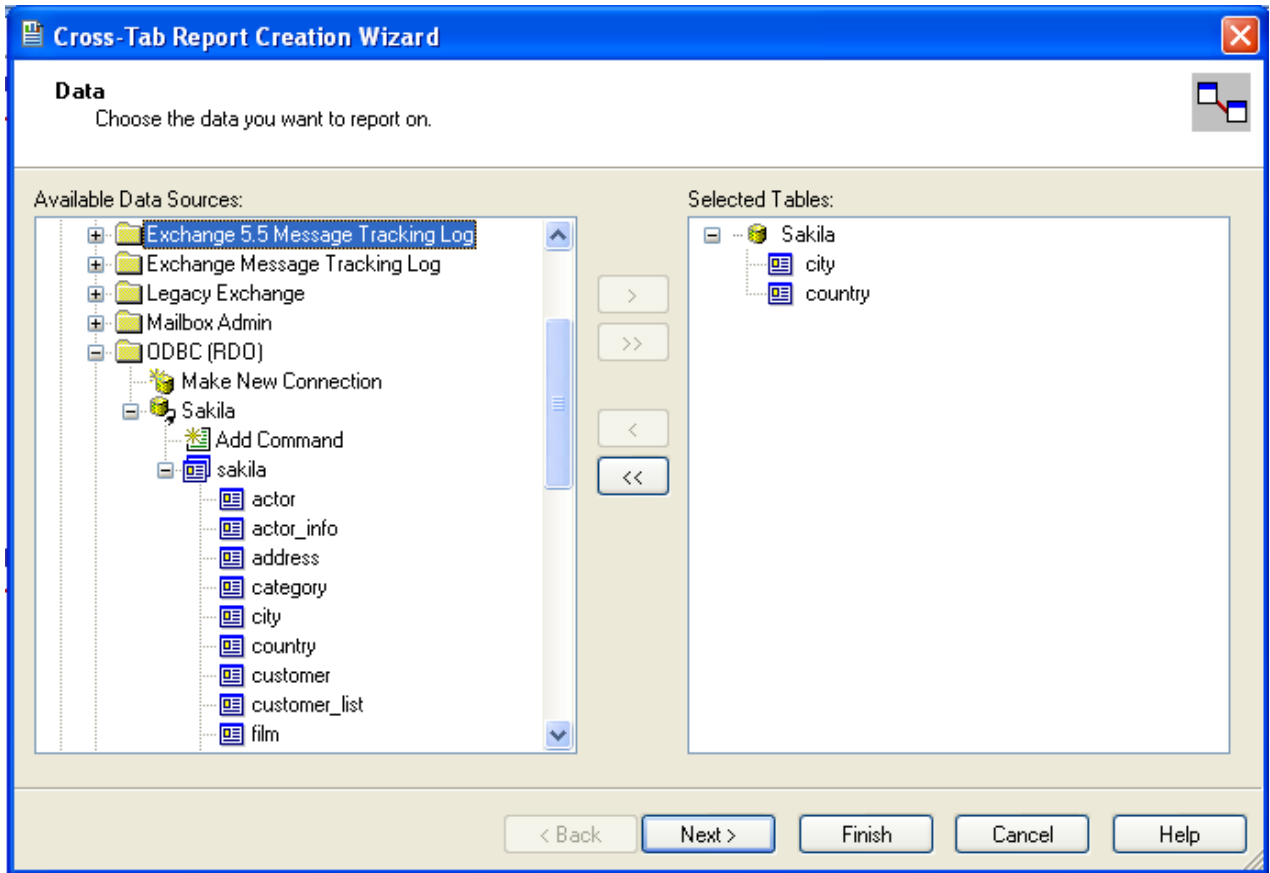
If the DSN contains a username/password combination, or you want to use different authentication credentials, click **NEXT** to enter the username and password that you want to use. Otherwise, click **FINISH** to continue the data source selection wizard.



5. You will be returned the Cross-Tab Report Creation Wizard. You now need to select the database and tables that you want to include in your report. For our example, we will expand the selected Sakila database. Click the **city** table and use the > button to add the table to the report. Then repeat the action with the **country** table. Alternatively you can select multiple tables and add them to the report.

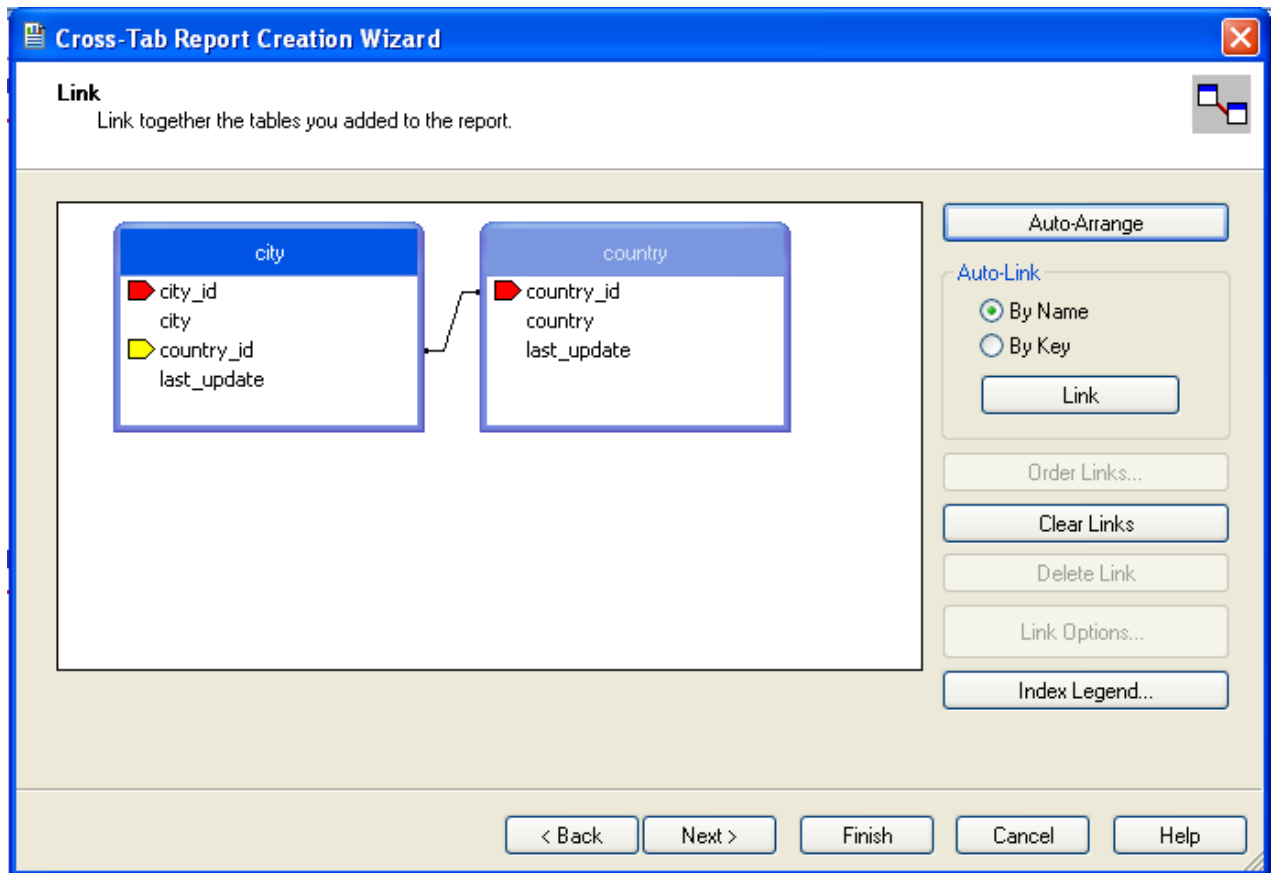
Finally, you can select the parent **SAKILA** resource and add of the tables to the report.

Once you have selected the tables you want to include, click **NEXT** to continue.



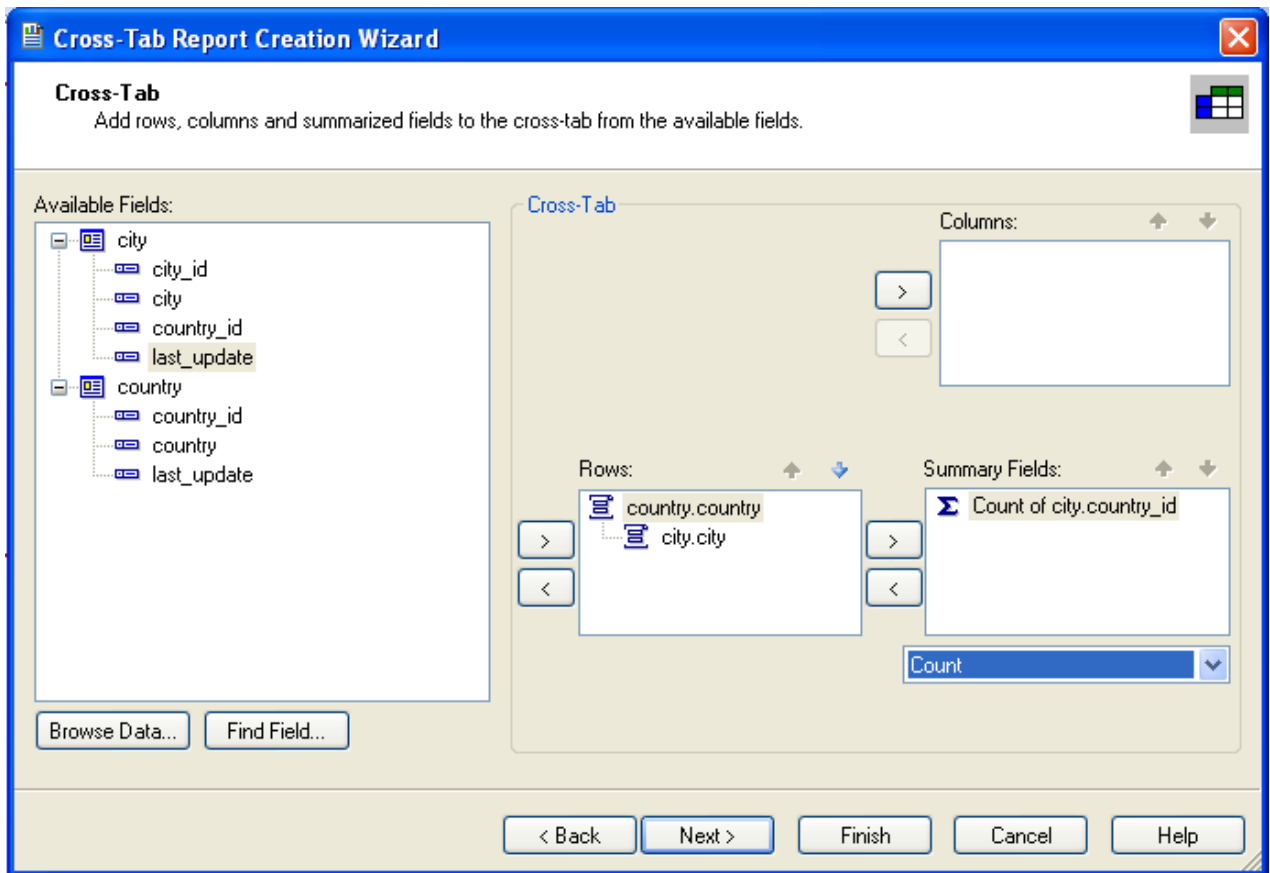
6. Crystal Reports will now read the table definitions and automatically identify the links between the tables. The identification of links between tables enables Crystal Reports to automatically lookup and summarize information based on all the tables in the database according to your query. If Crystal Reports is unable to perform the linking itself, you can manually create the links between fields in the tables you have selected.

Click NEXT to continue the process.



7. You can now select the columns and rows that you wish to include within the Cross-Tab report. Drag and drop or use the > buttons to add fields to each area of the report. In the example shown, we will report on cities, organized by country, incorporating a count of the number of cities within each country. If you want to browse the data, select a field and click the BROWSE DATA... button.

Click NEXT to create a graph of the results. Since we are not creating a graph from this data, click FINISH to generate the report.



8. The finished report will be shown, a sample of the output from the Sakila sample database is shown below.

		Total
Total		600
Afghanistan	Total	1
	Kabul	1
Algeria	Total	3
	Batna	1
	Bchar	1
	Skikda	1
American Samoa	Total	1
	Tafuna	1
Angola	Total	2
	Benguela	1
	Namibe	1
Anguilla	Total	1
	South Hill	1
Argentina	Total	13
	Almirante Brow	1

Once the ODBC connection has been opened within Crystal Reports, you can browse and add any fields within the available tables into your reports.

26.1.4.7. Connector/ODBC Programming

With a suitable ODBC Manager and the Connector/ODBC driver installed, any programming language or environment that can support ODBC should be able to connect to a MySQL database through Connector/ODBC.

This includes, but is certainly not limited to, Microsoft support languages (including Visual Basic, C# and interfaces such as ODBC.NET), Perl (through the DBI module, and the DBD::ODBC driver).

26.1.4.7.1. Using Connector/ODBC with Visual Basic Using ADO, DAO and RDO

This section contains simple examples of the use of MySQL ODBC 3.51 Driver with ADO, DAO and RDO.

26.1.4.7.1.1. ADO: `rs.addNew`, `rs.delete`, and `rs.update`

The following ADO (ActiveX Data Objects) example creates a table `my_ado` and demonstrates the use of `rs.addNew`, `rs.delete`, and `rs.update`.

```
Private Sub myodbc_ado_Click()

Dim conn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim fld As ADODB.Field
Dim sql As String

'connect to MySQL server using MySQL ODBC 3.51 Driver
Set conn = New ADODB.Connection
conn.ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};" _
& "SERVER=localhost;" _
& " DATABASE=test;" _
& "UID=venu;PWD=venu; OPTION=3"

conn.Open

'create table
conn.Execute "DROP TABLE IF EXISTS my_ado"
conn.Execute "CREATE TABLE my_ado(id int not null primary key, name varchar(20)," _
& "txt text, dt date, tm time, ts timestamp)"

'direct insert
conn.Execute "INSERT INTO my_ado(id,name,txt) values(1,100,'venu')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(2,200,'MySQL')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(3,300,'Delete')"

Set rs = New ADODB.Recordset
rs.CursorLocation = adUseServer

'fetch the initial table ..
rs.Open "SELECT * FROM my_ado", conn
Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Initial my_ado Result Set " & String(50, "-")
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print

Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close

'rs insert
rs.Open "select * from my_ado", conn, adOpenDynamic, adLockOptimistic
rs.AddNew
rs!Name = "Monty"
rs!txt = "Insert row"
rs.Update
rs.Close

'rs update
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-row"
rs.Update
rs.Close

'rs update second time..
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-second-time"
rs.Update
rs.Close

'rs delete
rs.Open "SELECT * FROM my_ado"
rs.MoveNext
rs.MoveNext
rs.Delete
rs.Close
```

```

'fetch the updated table ..
rs.Open "SELECT * FROM my_ado", conn
Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Updated my_ado Result Set " & String(50, "-")
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print

Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close
conn.Close
End Sub

```

26.1.4.7.1.2. DAO: **rs.addNew**, **rs.update**, and Scrolling

The following DAO (Data Access Objects) example creates a table **my_dao** and demonstrates the use of **rs.addNew**, **rs.update**, and result set scrolling.

```

Private Sub myodbc_dao_Click()

Dim ws As Workspace
Dim conn As Connection
Dim queryDef As queryDef
Dim str As String

'connect to MySQL using MySQL ODBC 3.51 Driver
Set ws = DBEngine.CreateWorkspace("", "venu", "venu", dbUseODBC)
str = "odbc;DRIVER={MySQL ODBC 3.51 Driver};" _
& "SERVER=localhost;" _
& " DATABASE=test;" _
& "UID=venu;PWD=venu; OPTION=3"
Set conn = ws.OpenConnection("test", dbDriverNoPrompt, False, str)

'Create table my_dao
Set queryDef = conn.CreateQueryDef("", "drop table if exists my_dao")
queryDef.Execute

Set queryDef = conn.CreateQueryDef("", "create table my_dao(Id INT AUTO_INCREMENT PRIMARY KEY, " _
& "Ts TIMESTAMP(14) NOT NULL, Name varchar(20), Id2 INT)")
queryDef.Execute

'Insert new records using rs.addNew
Set rs = conn.OpenRecordset("my_dao")
Dim i As Integer

For i = 10 To 15
rs.AddNew
rs!Name = "insert record" & i
rs!Id2 = i
rs.Update
Next i
rs.Close

'rs update..
Set rs = conn.OpenRecordset("my_dao")
rs.Edit
rs!Name = "updated-string"
rs.Update
rs.Close

'fetch the table back...
Set rs = conn.OpenRecordset("my_dao", dbOpenDynamic)
str = "Results:"
rs.MoveFirst
While Not rs.EOF
str = " " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print "DATA:" & str
rs.MoveNext
Wend

'rs Scrolling
rs.MoveFirst
str = " FIRST ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str

rs.MoveLast
str = " LAST ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str

```

```

rs.MovePrevious
str = " LAST-1 ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str

'free all resources
rs.Close
queryDef.Close
conn.Close
ws.Close

End Sub

```

26.1.4.7.1.3. RDO: `rs.addNew` and `rs.update`

The following RDO (Remote Data Objects) example creates a table `my_rdo` and demonstrates the use of `rs.addNew` and `rs.update`.

```

Dim rs As rdoResultset
Dim cn As New rdoConnection
Dim cl As rdoColumn
Dim SQL As String

'cn.Connect = "DSN=test;"
cn.Connect = "DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& " DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"

cn.CursorDriver = rdUseOdbc
cn.EstablishConnection rdDriverPrompt

'drop table my_rdo
SQL = "drop table if exists my_rdo"
cn.Execute SQL, rdExecDirect

'create table my_rdo
SQL = "create table my_rdo(id int, name varchar(20))"
cn.Execute SQL, rdExecDirect

'insert - direct
SQL = "insert into my_rdo values (100,'venu')"
cn.Execute SQL, rdExecDirect

SQL = "insert into my_rdo values (200,'MySQL')"
cn.Execute SQL, rdExecDirect

'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 300
rs!Name = "Insert1"
rs.Update
rs.Close

'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 400
rs!Name = "Insert 2"
rs.Update
rs.Close

'rs update
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.Edit
rs!id = 999
rs!Name = "updated"
rs.Update
rs.Close

'fetch back...
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
Do Until rs.EOF
For Each cl In rs.rdoColumns
Debug.Print cl.Value,
Next
rs.MoveNext
Debug.Print
Loop
Debug.Print "Row count="; rs.RowCount

```



```
'close
rs.Close
cn.Close

End Sub
```

26.1.4.7.2. Using Connector/ODBC with .NET

This section contains simple examples that demonstrate the use of Connector/ODBC drivers with ODBC.NET.

26.1.4.7.2.1. Using Connector/ODBC with ODBC.NET and C# (C sharp)

The following sample creates a table `my_odbc_net` and demonstrates its use in C#.

```
/**
 * @sample      : mycon.cs
 * @purpose     : Demo sample for ODBC.NET using Connector/ODBC
 * @author      : Venu, <myodbc@lists.mysql.com>
 *
 * (C) Copyright MySQL AB, 1995-2006
 *
 */

/* build command
 *
 * csc /t:exe
 *      /out:mycon.exe mycon.cs
 *      /r:Microsoft.Data.Odbc.dll
 */

using Console = System.Console;
using Microsoft.Data.Odbc;

namespace myodbc3
{
    class mycon
    {
        static void Main(string[] args)
        {
            try
            {
                //Connection string for MyODBC 2.50
                /*string MyConString = "DRIVER={MySQL};" +
                "SERVER=localhost;" +
                "DATABASE=test;" +
                "UID=venu;" +
                "PASSWORD=venu;" +
                "OPTION=3";
                */
                //Connection string for Connector/ODBC 3.51
                string MyConString = "DRIVER={MySQL ODBC 3.51 Driver};" +
                "SERVER=localhost;" +
                "DATABASE=test;" +
                "UID=venu;" +
                "PASSWORD=venu;" +
                "OPTION=3";

                //Connect to MySQL using Connector/ODBC
                OdbcConnection MyConnection = new OdbcConnection(MyConString);
                MyConnection.Open();

                Console.WriteLine("\n !!! success, connected successfully !!!\n");

                //Display connection information
                Console.WriteLine("Connection Information:");
                Console.WriteLine("\tConnection String:" +
                MyConnection.ConnectionString);
                Console.WriteLine("\tConnection Timeout:" +
                MyConnection.ConnectionTimeout);
                Console.WriteLine("\tDatabase:" +
                MyConnection.Database);
                Console.WriteLine("\tDataSource:" +
                MyConnection.DataSource);
                Console.WriteLine("\tDriver:" +
                MyConnection.Driver);
                Console.WriteLine("\tServerVersion:" +
                MyConnection.ServerVersion);

                //Create a sample table
                OdbcCommand MyCommand =
                new OdbcCommand("DROP TABLE IF EXISTS my_odbc_net",
                MyConnection);
                MyCommand.ExecuteNonQuery();
                MyCommand.CommandText =
```

```

"CREATE TABLE my_odbc_net(id int, name varchar(20), idb bigint)";
MyCommand.ExecuteNonQuery();

//Insert
MyCommand.CommandText =
"INSERT INTO my_odbc_net VALUES(10,'venu', 300)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());

//Insert
MyCommand.CommandText =
"INSERT INTO my_odbc_net VALUES(20,'mysql',400)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());

//Insert
MyCommand.CommandText =
"INSERT INTO my_odbc_net VALUES(20,'mysql',500)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());

//Update
MyCommand.CommandText =
"UPDATE my_odbc_net SET id=999 WHERE id=20";
Console.WriteLine("Update, Total rows affected:" +
    MyCommand.ExecuteNonQuery());

//COUNT(*)
MyCommand.CommandText =
"SELECT COUNT(*) as TRows FROM my_odbc_net";
Console.WriteLine("Total Rows:" +
    MyCommand.ExecuteScalar());

//Fetch
MyCommand.CommandText = "SELECT * FROM my_odbc_net";
OdbcDataReader MyDataReader;
MyDataReader = MyCommand.ExecuteReader();
while (MyDataReader.Read())
{
    if(string.Compare(MyConnection.Driver,"myodbc3.dll") == 0) {
        //Supported only by Connector/ODBC 3.51
        Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
            MyDataReader.GetString(1) + " " +
            MyDataReader.GetInt64(2));
    }
    else {
        //BIGINTs not supported by Connector/ODBC
        Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
            MyDataReader.GetString(1) + " " +
            MyDataReader.GetInt32(2));
    }
}

//Close all resources
MyDataReader.Close();
MyConnection.Close();
}
catch (OdbcException MyOdbcException) //Catch any ODBC exception ..
{
    for (int i=0; i < MyOdbcException.Errors.Count; i++)
    {
        Console.WriteLine("ERROR #" + i + "\n" +
            "Message: " +
            MyOdbcException.Errors[i].Message + "\n" +
            "Native: " +
            MyOdbcException.Errors[i].NativeError.ToString() + "\n" +
            "Source: " +
            MyOdbcException.Errors[i].Source + "\n" +
            "SQL: " +
            MyOdbcException.Errors[i].SQLState + "\n");
    }
}
}
}
}
}

```

26.1.4.7.2.2. Using Connector/ODBC with ODBC.NET and Visual Basic

The following sample creates a table `my_vb_net` and demonstrates the use in VB.

```

' @sample      : myvb.vb
' @purpose     : Demo sample for ODBC.NET using Connector/ODBC
' @author      : Venu, <myodbc@lists.mysql.com>
'
' (C) Copyright MySQL AB, 1995-2006
'

```

```

' build command
'
' vbc /target:exe
'      /out:myvb.exe
'      /r:Microsoft.Data.Odbc.dll
'      /r:System.dll
'      /r:System.Data.dll
'
Imports Microsoft.Data.Odbc
Imports System

Module myvb
    Sub Main()
        Try

            'Connector/ODBC 3.51 connection string
            Dim MyConString As String = "DRIVER={MySQL ODBC 3.51 Driver};" & _
                "SERVER=localhost;" & _
                "DATABASE=test;" & _
                "UID=venu;" & _
                "PASSWORD=venu;" & _
                "OPTION=3;"

            'Connection
            Dim MyConnection As New OdbcConnection(MyConString)
            MyConnection.Open()

            Console.WriteLine("Connection State:" & MyConnection.State.ToString)

            'Drop
            Console.WriteLine("Dropping table")
            Dim MyCommand As New OdbcCommand()
            MyCommand.Connection = MyConnection
            MyCommand.CommandText = "DROP TABLE IF EXISTS my_vb_net"
            MyCommand.ExecuteNonQuery()

            'Create
            Console.WriteLine("Creating...")
            MyCommand.CommandText = "CREATE TABLE my_vb_net(id int, name varchar(30))"
            MyCommand.ExecuteNonQuery()

            'Insert
            MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(10,'venu')"
            Console.WriteLine("INSERT, Total rows affected:" & _
                MyCommand.ExecuteNonQuery())

            'Insert
            MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
            Console.WriteLine("INSERT, Total rows affected:" & _
                MyCommand.ExecuteNonQuery())

            'Insert
            MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
            Console.WriteLine("INSERT, Total rows affected:" & _
                MyCommand.ExecuteNonQuery())

            'Insert
            MyCommand.CommandText = "INSERT INTO my_vb_net(id) VALUES(30)"
            Console.WriteLine("INSERT, Total rows affected:" & _
                MyCommand.ExecuteNonQuery())

            'Update
            MyCommand.CommandText = "UPDATE my_vb_net SET id=999 WHERE id=20"
            Console.WriteLine("Update, Total rows affected:" & _
                MyCommand.ExecuteNonQuery())

            'COUNT(*)
            MyCommand.CommandText = "SELECT COUNT(*) as TRows FROM my_vb_net"
            Console.WriteLine("Total Rows:" & MyCommand.ExecuteScalar())

            'Select
            Console.WriteLine("Select * FROM my_vb_net")
            MyCommand.CommandText = "SELECT * FROM my_vb_net"
            Dim MyDataReader As OdbcDataReader
            MyDataReader = MyCommand.ExecuteReader
            While MyDataReader.Read
                If MyDataReader("name") Is DBNull.Value Then
                    Console.WriteLine("id = " & _
                        CStr(MyDataReader("id")) & " name = " & _
                        "NULL")
                Else
                    Console.WriteLine("id = " & _
                        CStr(MyDataReader("id")) & " name = " & _
                        CStr(MyDataReader("name")))
                End If
            End While

            'Catch ODBC Exception
            Catch MyOdbcException As OdbcException

```

```

Dim i As Integer
Console.WriteLine(MyOdbcException.ToString)

'Catch program exception
Catch MyException As Exception
    Console.WriteLine(MyException.ToString)
End Try
End Sub

```

26.1.5. Connector/ODBC Reference

This section provides reference material for the Connector/ODBC API, showing supported functions and methods, supported MySQL column types and the corresponding native type in Connector/ODBC, and the error codes returned by Connector/ODBC when a fault occurs.

26.1.5.1. Connector/ODBC API Reference

This section summarizes ODBC routines, categorized by functionality.

For the complete ODBC API reference, please refer to the ODBC Programmer's Reference at http://msdn.microsoft.com/library/en-us/odbc/htm/odbcabout_this_manual.asp.

An application can call [SQLGetInfo](#) function to obtain conformance information about Connector/ODBC. To obtain information about support for a specific function in the driver, an application can call [SQLGetFunctions](#).

Note

For backward compatibility, the Connector/ODBC 3.51 driver supports all deprecated functions.

The following tables list Connector/ODBC API calls grouped by task:

Connecting to a data source:

Function name	Connector/ ODBC		Standard	Purpose
	2.50	3.51		
SQLAllocHandle	No	Yes	ISO 92	Obtains an environment, connection, statement, or descriptor handle.
SQLConnect	Yes	Yes	ISO 92	Connects to a specific driver by data source name, user ID, and password.
SQLDriverConnect	Yes	Yes	ODBC	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialog boxes for the user.
SQLAllocEnv	Yes	Yes	Deprecated	Obtains an environment handle allocated from driver.
SQLAllocConnect	Yes	Yes	Deprecated	Obtains a connection handle

Obtaining information about a driver and data source:

Function name	Connector/ ODBC		Standard	Purpose
	2.50	3.51		
SQLDataSources	No	No	ISO 92	Returns the list of available data sources, handled by the Driver Manager
SQLDrivers	No	No	ODBC	Returns the list of installed drivers and their attributes, handles by Driver Manager
SQLGetInfo	Yes	Yes	ISO 92	Returns information about a specific driver and data source.
SQLGetFunctions	Yes	Yes	ISO 92	Returns supported driver functions.
SQLGetTypeInfo	Yes	Yes	ISO 92	Returns information about supported data types.

Setting and retrieving driver attributes:

Function name	Connector/ ODBC		Standard	Purpose
	2.50	3.51		
SQLSetConnectAttr	No	Yes	ISO 92	Sets a connection attribute.
SQLGetConnectAttr	No	Yes	ISO 92	Returns the value of a connection attribute.
SQLSetConnectOption	Yes	Yes	Deprecated	Sets a connection option
SQLGetConnectOption	Yes	Yes	Deprecated	Returns the value of a connection option
SQLSetEnvAttr	No	Yes	ISO 92	Sets an environment attribute.
SQLGetEnvAttr	No	Yes	ISO 92	Returns the value of an environment attribute.
SQLSetStmtAttr	No	Yes	ISO 92	Sets a statement attribute.
SQLGetStmtAttr	No	Yes	ISO 92	Returns the value of a statement attribute.
SQLSetStmtOption	Yes	Yes	Deprecated	Sets a statement option
SQLGetStmtOption	Yes	Yes	Deprecated	Returns the value of a statement option

Preparing SQL requests:

Function name	Connector/ ODBC		Standard	Purpose
	2.50	3.51		
SQLAllocStmt	Yes	Yes	Deprecated	Allocates a statement handle
SQLPrepare	Yes	Yes	ISO 92	Prepares an SQL statement for later execution.
SQLBindParameter	Yes	Yes	ODBC	Assigns storage for a parameter in an SQL statement.
SQLGetCursorName	Yes	Yes	ISO 92	Returns the cursor name associated with a statement handle.
SQLSetCursorName	Yes	Yes	ISO 92	Specifies a cursor name.
SQLSetScrollOptions	Yes	Yes	ODBC	Sets options that control cursor behavior.

Submitting requests:

Function name	Connector/ ODBC		Standard	Purpose
	2.50	3.51		
SQLExecute	Yes	Yes	ISO 92	Executes a prepared statement.
SQLExecDirect	Yes	Yes	ISO 92	Executes a statement
SQLNativeSql	Yes	Yes	ODBC	Returns the text of an SQL statement as translated by the driver.
SQLDescribeParam	Yes	Yes	ODBC	Returns the description for a specific parameter in a statement.
SQLNumParams	Yes	Yes	ISO 92	Returns the number of parameters in a statement.
SQLParamData	Yes	Yes	ISO 92	Used in conjunction with SQLPutData to supply parameter data at execution time. (Useful for long data values.)
SQLPutData	Yes	Yes	ISO 92	Sends part or all of a data value for a parameter. (Useful for long data values.)

Retrieving results and information about results:

Function name	Connector/ ODBC		Standard	Purpose
	2.50	3.51		

SQLRowCount	Yes	Yes	ISO 92	Returns the number of rows affected by an insert, update, or delete request.
SQLNumResultCols	Yes	Yes	ISO 92	Returns the number of columns in the result set.
SQLDescribeCol	Yes	Yes	ISO 92	Describes a column in the result set.
SQLColAttribute	No	Yes	ISO 92	Describes attributes of a column in the result set.
SQLColAttributes	Yes	Yes	Deprecated	Describes attributes of a column in the result set.
SQLFetch	Yes	Yes	ISO 92	Returns multiple result rows.
SQLFetchScroll	No	Yes	ISO 92	Returns scrollable result rows.
SQLExtendedFetch	Yes	Yes	Deprecated	Returns scrollable result rows.
SQLSetPos	Yes	Yes	ODBC	Positions a cursor within a fetched block of data and allows an application to refresh data in the rowset or to update or delete data in the result set.
SQLBulkOperations	No	Yes	ODBC	Performs bulk insertions and bulk bookmark operations, including update, delete, and fetch by bookmark.

Retrieving error or diagnostic information:

Function name	Connector/ ODBC		Standard	Purpose
	2.50	3.51		
SQLError	Yes	Yes	Deprecated	Returns additional error or status information
SQLGetDiagField	Yes	Yes	ISO 92	Returns additional diagnostic information (a single field of the diagnostic data structure).
SQLGetDiagRec	Yes	Yes	ISO 92	Returns additional diagnostic information (multiple fields of the diagnostic data structure).

Obtaining information about the data source's system tables (catalog functions) item:

Function name	Connector/ ODBC		Standard	Purpose
	2.50	3.51		
SQLColumnPrivileges	Yes	Yes	ODBC	Returns a list of columns and associated privileges for one or more tables.
SQLColumns	Yes	Yes	X/Open	Returns the list of column names in specified tables.
SQLForeignKeys	Yes	Yes	ODBC	Returns a list of column names that make up foreign keys, if they exist for a specified table.
SQLPrimaryKeys	Yes	Yes	ODBC	Returns the list of column names that make up the primary key for a table.
SQLSpecialColumns	Yes	Yes	X/Open	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
SQLStatistics	Yes	Yes	ISO 92	Returns statistics about a single table and the list of indexes associated with the table.
SQLTablePrivileges	Yes	Yes	ODBC	Returns a list of tables and the privileges associated with each table.
SQLTables	Yes	Yes	X/Open	Returns the list of table names stored in a specific data source.

Performing transactions:

	Connector/		

Function name	ODBC		Standard	Purpose
	2.50	3.51		
SQLTransact	Yes	Yes	Deprecated	Commits or rolls back a transaction
SQLEndTran	No	Yes	ISO 92	Commits or rolls back a transaction.

Terminating a statement:

Function name	Connector/ ODBC		Standard	Purpose
	2.50	3.51		
SQLFreeStmt	Yes	Yes	ISO 92	Ends statement processing, discards pending results, and, optionally, frees all resources associated with the statement handle.
SQLCloseCursor	Yes	Yes	ISO 92	Closes a cursor that has been opened on a statement handle.
SQLCancel	Yes	Yes	ISO 92	Cancels an SQL statement.

Terminating a connection:

Function name	Connector/ ODBC		Standard	Purpose
	2.50	3.51		
SQLDisconnect	Yes	Yes	ISO 92	Closes the connection.
SQLFreeHandle	No	Yes	ISO 92	Releases an environment, connection, statement, or descriptor handle.
SQLFreeConnect	Yes	Yes	Deprecated	Releases connection handle
SQLFreeEnv	Yes	Yes	Deprecated	Releases an environment handle

26.1.5.2. Connector/ODBC Data Types

The following table illustrates how driver maps the server data types to default SQL and C data types:

Native Value	SQL Type	C Type
bit	SQL_BIT	SQL_C_BIT
tinyint	SQL_TINYINT	SQL_C_STINYINT
tinyint unsigned	SQL_TINYINT	SQL_C_UTINYINT
bigint	SQL_BIGINT	SQL_C_SBIGINT
bigint unsigned	SQL_BIGINT	SQL_C_UBIGINT
long varbinary	SQL_LONGVARBINARY	SQL_C_BINARY
blob	SQL_LONGVARBINARY	SQL_C_BINARY
longblob	SQL_LONGVARBINARY	SQL_C_BINARY
tinyblob	SQL_LONGVARBINARY	SQL_C_BINARY
mediumblob	SQL_LONGVARBINARY	SQL_C_BINARY
long varchar	SQL_LONGVARCHAR	SQL_C_CHAR
text	SQL_LONGVARCHAR	SQL_C_CHAR
mediumtext	SQL_LONGVARCHAR	SQL_C_CHAR
char	SQL_CHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_CHAR
decimal	SQL_DECIMAL	SQL_C_CHAR
integer	SQL_INTEGER	SQL_C_SLONG

integer unsigned	SQL_INTEGER	SQL_C_ULONG
int	SQL_INTEGER	SQL_C_SLONG
int unsigned	SQL_INTEGER	SQL_C_ULONG
mediumint	SQL_INTEGER	SQL_C_SLONG
mediumint unsigned	SQL_INTEGER	SQL_C_ULONG
smallint	SQL_SMALLINT	SQL_C_SSHORT
smallint unsigned	SQL_SMALLINT	SQL_C_USHORT
real	SQL_FLOAT	SQL_C_DOUBLE
double	SQL_FLOAT	SQL_C_DOUBLE
float	SQL_REAL	SQL_C_FLOAT
double precision	SQL_DOUBLE	SQL_C_DOUBLE
date	SQL_DATE	SQL_C_DATE
time	SQL_TIME	SQL_C_TIME
year	SQL_SMALLINT	SQL_C_SHORT
datetime	SQL_TIMESTAMP	SQL_C_TIMESTAMP
timestamp	SQL_TIMESTAMP	SQL_C_TIMESTAMP
text	SQL_VARCHAR	SQL_C_CHAR
varchar	SQL_VARCHAR	SQL_C_CHAR
enum	SQL_VARCHAR	SQL_C_CHAR
set	SQL_VARCHAR	SQL_C_CHAR
bit	SQL_CHAR	SQL_C_CHAR
bool	SQL_CHAR	SQL_C_CHAR

26.1.5.3. Connector/ODBC Error Codes

The following tables lists the error codes returned by the driver apart from the server errors.

Native Code	SQLSTATE 2	SQLSTATE 3	Error Message
500	01000	01000	General warning
501	01004	01004	String data, right truncated
502	01S02	01S02	Option value changed
503	01S03	01S03	No rows updated/deleted
504	01S04	01S04	More than one row updated/deleted
505	01S06	01S06	Attempt to fetch before the result set returned the first row set
506	07001	07002	SQLBindParameter not used for all parameters
507	07005	07005	Prepared statement not a cursor-specification
508	07009	07009	Invalid descriptor index
509	08002	08002	Connection name in use
510	08003	08003	Connection does not exist
511	24000	24000	Invalid cursor state
512	25000	25000	Invalid transaction state
513	25S01	25S01	Transaction state unknown
514	34000	34000	Invalid cursor name
515	S1000	HY000	General driver defined error
516	S1001	HY001	Memory allocation error

517	S1002	HY002	Invalid column number
518	S1003	HY003	Invalid application buffer type
519	S1004	HY004	Invalid SQL data type
520	S1009	HY009	Invalid use of null pointer
521	S1010	HY010	Function sequence error
522	S1011	HY011	Attribute can not be set now
523	S1012	HY012	Invalid transaction operation code
524	S1013	HY013	Memory management error
525	S1015	HY015	No cursor name available
526	S1024	HY024	Invalid attribute value
527	S1090	HY090	Invalid string or buffer length
528	S1091	HY091	Invalid descriptor field identifier
529	S1092	HY092	Invalid attribute/option identifier
530	S1093	HY093	Invalid parameter number
531	S1095	HY095	Function type out of range
532	S1106	HY106	Fetch type out of range
533	S1117	HY117	Row value out of range
534	S1109	HY109	Invalid cursor position
535	S1C00	HYC00	Optional feature not implemented
0	21S01	21S01	Column count does not match value count
0	23000	23000	Integrity constraint violation
0	42000	42000	Syntax error or access violation
0	42S02	42S02	Base table or view not found
0	42S12	42S12	Index not found
0	42S21	42S21	Column already exists
0	42S22	42S22	Column not found
0	08S01	08S01	Communication link failure

26.1.6. Connector/ODBC Notes and Tips

Here are some common notes and tips for using Connector/ODBC within different environments, applications and tools. The notes provided here are based on the experiences of Connector/ODBC developers and users.

26.1.6.1. Connector/ODBC General Functionality

This section provides help with common queries and areas of functionality in MySQL and how to use them with Connector/ODBC.

26.1.6.1.1. Obtaining Auto-Increment Values

Obtaining the value of column that uses [AUTO_INCREMENT](#) after an [INSERT](#) statement can be achieved in a number of different ways. To obtain the value immediately after an [INSERT](#), use a [SELECT](#) query with the [LAST_INSERT_ID\(\)](#) function.

For example, using Connector/ODBC you would execute two separate statements, the [INSERT](#) statement and the [SELECT](#) query to obtain the auto-increment value.

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

If you do not require the value within your application, but do require the value as part of another [INSERT](#), the entire process can be handled by executing the following statements:

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');  
INSERT INTO tbl2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

Certain ODBC applications (including Delphi and Access) may have trouble obtaining the auto-increment value using the previous examples. In this case, try the following statement as an alternative:

```
SELECT * FROM tbl WHERE auto IS NULL;
```

See [Section 25.2.14.3, “How to Get the Unique ID for the Last Inserted Row”](#).

26.1.6.1.2. Dynamic Cursor Support

Support for the [dynamic cursor](#) is provided in Connector/ODBC 3.51, but dynamic cursors are not enabled by default. You can enable this function within Windows by selecting the [Enable Dynamic Cursor](#) checkbox within the ODBC Data Source Administrator.

On other platforms, you can enable the dynamic cursor by adding [32](#) to the [OPTION](#) value when creating the DSN.

26.1.6.1.3. Connector/ODBC Performance

The Connector/ODBC driver has been optimized to provide very fast performance. If you experience problems with the performance of Connector/ODBC, or notice a large amount of disk activity for simple queries, there are a number of aspects you should check:

- Ensure that [ODBC Tracing](#) is not enabled. With tracing enabled, a lot of information is recorded in the tracing file by the ODBC Manager. You can check, and disable, tracing within Windows using the TRACING panel of the ODBC Data Source Administrator. Within Mac OS X, check the TRACING panel of ODBC Administrator. See [Section 26.1.3.8, “Getting an ODBC Trace File”](#).
- Make sure you are using the standard version of the driver, and not the debug version. The debug version includes additional checks and reporting measures.
- Disable the Connector/ODBC driver trace and query logs. These options are enabled for each DSN, so make sure to examine only the DSN that you are using in your application. Within Windows, you can disable the Connector/ODBC and query logs by modifying the DSN configuration. Within Mac OS X and Unix, ensure that the driver trace (option value 4) and query logging (option value 524288) are not enabled.

26.1.6.1.4. Setting ODBC Query Timeout in Windows

For more information on how to set the query timeout on Microsoft Windows when executing queries through an ODBC connection, read the Microsoft knowledgebase document at <http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B153756>.

26.1.6.2. Connector/ODBC Application Specific Tips

Most programs should work with Connector/ODBC, but for each of those listed here, there are specific notes and tips to improve or enhance the way you work with Connector/ODBC and these applications.

With all applications you should ensure that you are using the latest Connector/ODBC drivers, ODBC Manager and any supporting libraries and interfaces used by your application. For example, on Windows, using the latest version of Microsoft Data Access Components (MDAC) will improve the compatibility with ODBC in general, and with the Connector/ODBC driver.

26.1.6.2.1. Using Connector/ODBC with Microsoft Applications

The majority of Microsoft applications have been tested with Connector/ODBC, including Microsoft Office, Microsoft Access and the various programming languages supported within ASP and Microsoft Visual Studio.

26.1.6.2.1.1. Microsoft Access

To improve the integration between Microsoft Access and MySQL through Connector/ODBC:

- For all versions of Access, you should enable the Connector/ODBC [Return matching rows](#) option. For Access 2.0, you should additionally enable the [Simulate ODBC 1.0](#) option.

- You should have a **TIMESTAMP** column in all tables that you want to be able to update. For maximum portability, don't use a length specification in the column declaration (which is unsupported within MySQL in versions earlier than 4.1).
- You should have a primary key in each MySQL table you want to use with Access. If not, new or updated rows may show up as **#DELETED#**.
- Use only **DOUBLE** float fields. Access fails when comparing with single-precision floats. The symptom usually is that new or updated rows may show up as **#DELETED#** or that you can't find or update rows.
- If you are using Connector/ODBC to link to a table that has a **BIGINT** column, the results are displayed as **#DELETED#**. The work around solution is:
 - Have one more dummy column with **TIMESTAMP** as the data type.
 - Select the **Change BIGINT columns to INT** option in the connection dialog in ODBC DSN Administrator.
 - Delete the table link from Access and re-create it.

Old records may still display as **#DELETED#**, but newly added/updated records are displayed properly.

- If you still get the error **Another user has changed your data** after adding a **TIMESTAMP** column, the following trick may help you:

Don't use a **table** data sheet view. Instead, create a form with the fields you want, and use that **form** data sheet view. You should set the **DefaultValue** property for the **TIMESTAMP** column to **NOW()**. It may be a good idea to hide the **TIMESTAMP** column from view so your users are not confused.
- In some cases, Access may generate SQL statements that MySQL can't understand. You can fix this by selecting **"Query|SQLSpecific|Pass-Through"** from the Access menu.
- On Windows NT, Access reports **BLOB** columns as **OLE OBJECTS**. If you want to have **MEMO** columns instead, you should change **BLOB** columns to **TEXT** with **ALTER TABLE**.
- Access can't always handle the MySQL **DATE** column properly. If you have a problem with these, change the columns to **DATE - TIME**.
- If you have in Access a column defined as **BYTE**, Access tries to export this as **TINYINT** instead of **TINYINT UNSIGNED**. This gives you problems if you have values larger than 127 in the column.
- If you have very large (long) tables in Access, it might take a very long time to open them. Or you might run low on virtual memory and eventually get an **ODBC Query Failed** error and the table cannot open. To deal with this, select the following options:
 - Return Matching Rows (2)
 - Allow BIG Results (8).

These add up to a value of 10 (**OPTION=10**).

Some external articles and tips that may be useful when using Access, ODBC and Connector/ODBC:

- Read [How to Trap ODBC Login Error Messages in Access](#)
- Optimizing Access ODBC Applications
 - [Optimizing for Client/Server Performance](#)
 - [Tips for Converting Applications to Using ODBCDirect](#)
 - [Tips for Optimizing Queries on Attached SQL Tables](#)
- For a list of tools that can be used with Access and ODBC data sources, refer to [converters](#) section for list of available tools.

MySQL Enterprise subscribers will find more information about using ODBC with Access in Knowledge Base articles such as [Use MySQL-Specific Syntax with Microsoft Access](#). To subscribe to MySQL Enterprise see <http://www.mysql.com/products/enterprise/advisors.html>.

26.1.6.2.1.2. Microsoft Excel and Column Types

If you have problems importing data into Microsoft Excel, particularly numerical, date, and time values, this is probably because of a bug in Excel, where the column type of the source data is used to determine the data type when that data is inserted into a cell within the worksheet. The result is that Excel incorrectly identifies the content and this affects both the display format and the data when it is used within calculations.

To address this issue, use the `CONCAT()` function in your queries. The use of `CONCAT()` forces Excel to treat the value as a string, which Excel will then parse and usually correctly identify the embedded information.

However, even with this option, some data may be incorrectly formatted, even though the source data remains unchanged. Use the `Format Cells` option within Excel to change the format of the displayed information.

26.1.6.2.1.3. Microsoft Visual Basic

To be able to update a table, you must define a primary key for the table.

Visual Basic with ADO can't handle big integers. This means that some queries like `SHOW PROCESSLIST` do not work properly. The fix is to use `OPTION=16384` in the ODBC connect string or to select the `Change BIGINT columns to INT` option in the Connector/ODBC connect screen. You may also want to select the `Return matching rows` option.

MySQL Enterprise

MySQL Enterprise subscribers can find a discussion about using VBA in the Knowledge Base article, [MySQL-Specific Syntax with VBA](#). To subscribe to MySQL Enterprise see <http://www.mysql.com/products/enterprise/advisors.html>.

26.1.6.2.1.4. Microsoft Visual InterDev

If you have a `BIGINT` in your result, you may get the error `[Microsoft][ODBC Driver Manager] Driver does not support this parameter`. Try selecting the `Change BIGINT columns to INT` option in the Connector/ODBC connect screen.

26.1.6.2.1.5. Visual Objects

You should select the `Don't optimize column widths` option.

26.1.6.2.1.6. Microsoft ADO

When you are coding with the ADO API and Connector/ODBC, you need to pay attention to some default properties that aren't supported by the MySQL server. For example, using the `CursorLocation Property` as `adUseServer` returns a result of `-1` for the `RecordCount Property`. To have the right value, you need to set this property to `adUseClient`, as shown in the VB code here:

```
Dim myconn As New ADODB.Connection
Dim myrs As New Recordset
Dim mySQL As String
Dim myrows As Long

myconn.Open "DSN=MyODBCsample"
mySQL = "SELECT * from user"
myrs.Source = mySQL
Set myrs.ActiveConnection = myconn
myrs.CursorLocation = adUseClient
myrs.Open
myrows = myrs.RecordCount

myrs.Close
myconn.Close
```

Another workaround is to use a `SELECT COUNT(*)` statement for a similar query to get the correct row count.

To find the number of rows affected by a specific SQL statement in ADO, use the `RecordsAffected` property in the ADO execute method. For more information on the usage of execute method, refer to <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ado270/html/mdmthcnexecute.asp>.

For information, see [ActiveX Data Objects\(ADO\) Frequently Asked Questions](#).

26.1.6.2.1.7. Using Connector/ODBC with Active Server Pages (ASP)

You should select the [Return matching rows](#) option in the DSN.

For more information about how to access MySQL via ASP using Connector/ODBC, refer to the following articles:

- [Using MyODBC To Access Your MySQL Database Via ASP](#)
- [ASP and MySQL at DWAM.NT](#)

A Frequently Asked Questions list for ASP can be found at <http://support.microsoft.com/default.aspx?scid=/Support/ActiveServer/faq/data/adofaq.asp>.

26.1.6.2.1.8. Using Connector/ODBC with Visual Basic (ADO, DAO and RDO) and ASP

Some articles that may help with Visual Basic and ASP:

- [MySQL BLOB columns and Visual Basic 6](#) by Mike Hillyer (<mike@openwin.org>).
- [How to map Visual basic data type to MySQL types](#) by Mike Hillyer (<mike@openwin.org>).

26.1.6.2.2. Using Connector/ODBC with Borland Applications

With all Borland applications where the Borland Database Engine (BDE) is used, follow these steps to improve compatibility:

- Update to BDE 3.2 or newer.
- Enable the [Don't optimize column widths](#) option in the DSN.
- Enabled the [Return matching rows](#) option in the DSN.

26.1.6.2.2.1. Using Connector/ODBC with Borland Builder 4

When you start a query, you can use the [Active](#) property or the [Open](#) method. Note that [Active](#) starts by automatically issuing a `SELECT * FROM ...` query. That may not be a good thing if your tables are large.

26.1.6.2.2.2. Using Connector/ODBC with Delphi

Also, here is some potentially useful Delphi code that sets up both an ODBC entry and a BDE entry for Connector/ODBC. The BDE entry requires a BDE Alias Editor that is free at a Delphi Super Page near you. (Thanks to Bryan Brunton <bryan@flesherfab.com> for this):

```
fReg:= TRegistry.Create;
fReg.OpenKey('\Software\ODBC\ODBC.INI\DocumentsFab', True);
fReg.WriteString('Database', 'Documents');
fReg.WriteString('Description', '');
fReg.WriteString('Driver', 'C:\WINNT\System32\myodbc.dll');
fReg.WriteString('Flag', '1');
fReg.WriteString('Password', '');
fReg.WriteString('Port', '');
fReg.WriteString('Server', 'xmark');
fReg.WriteString('User', 'winuser');
fReg.OpenKey('\Software\ODBC\ODBC.INI\ODBC Data Sources', True);
fReg.WriteString('DocumentsFab', 'MySQL');
fReg.CloseKey;
fReg.Free;

Memo1.Lines.Add('DATABASE NAME=');
Memo1.Lines.Add('USER NAME=');
Memo1.Lines.Add('ODBC DSN=DocumentsFab');
Memo1.Lines.Add('OPEN MODE=READ/WRITE');
Memo1.Lines.Add('BATCH COUNT=200');
Memo1.Lines.Add('LANGDRIVER=');
Memo1.Lines.Add('MAX ROWS=-1');
Memo1.Lines.Add('SCHEMA CACHE DIR=');
Memo1.Lines.Add('SCHEMA CACHE SIZE=8');
Memo1.Lines.Add('SCHEMA CACHE TIME=-1');
```

```
Memo1.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
Memo1.Lines.Add('SQLQRYMODE=');
Memo1.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
Memo1.Lines.Add('ENABLE BCD=FALSE');
Memo1.Lines.Add('ROWSET SIZE=20');
Memo1.Lines.Add('BLOBS TO CACHE=64');
Memo1.Lines.Add('BLOB SIZE=32');
AliasEditor.Add('DocumentsFab', 'MySQL', Memo1.Lines);
```

26.1.6.2.2.3. Using Connector/ODBC with C++ Builder

Tested with BDE 3.0. The only known problem is that when the table schema changes, query fields are not updated. BDE, however, does not seem to recognize primary keys, only the index named **PRIMARY**, although this has not been a problem.

26.1.6.2.3. Using Connector/ODBC with ColdFusion

The following information is taken from the ColdFusion documentation:

Use the following information to configure ColdFusion Server for Linux to use the **unixODBC** driver with Connector/ODBC for MySQL data sources. Allaire has verified that MyODBC 2.50.26 works with MySQL 3.22.27 and ColdFusion for Linux. (Any newer version should also work.) You can download Connector/ODBC at <http://dev.mysql.com/downloads/connector/odbc/>.

ColdFusion version 4.5.1 allows you to use the ColdFusion Administrator to add the MySQL data source. However, the driver is not included with ColdFusion version 4.5.1. Before the MySQL driver appears in the ODBC data sources drop-down list, you must build and copy the Connector/ODBC driver to `/opt/coldfusion/lib/libmyodbc.so`.

The Contrib directory contains the program `mysdn-xxx.zip` which allows you to build and remove the DSN registry file for the Connector/ODBC driver on ColdFusion applications.

For more information and guides on using ColdFusion and Connector/ODBC, see the following external sites:

- [Troubleshooting Data Sources and Database Connectivity for Unix Platforms](#).

26.1.6.2.4. Using Connector/ODBC with OpenOffice

Open Office (<http://www.openoffice.org>) [How-to: MySQL + OpenOffice](#). [How-to: OpenOffice + MyODBC + unixODBC](#).

26.1.6.2.5. Using Connector/ODBC with Sambar Server

Sambar Server (<http://www.sambarserver.info>) [How-to: MyODBC + SambarServer + MySQL](#).

26.1.6.2.6. Using Connector/ODBC with Pervasive Software DataJunction

You have to change it to output **VARCHAR** rather than **ENUM**, as it exports the latter in a manner that causes MySQL problems.

26.1.6.2.7. Using Connector/ODBC with SunSystems Vision

You should select the **Return matching rows** option.

26.1.6.3. Connector/ODBC Errors and Resolutions (FAQ)

The following section details some common errors and their suggested fix or alternative solution. If you are still experiencing problems, use the Connector/ODBC mailing list; see [Section 26.1.7.1, “Connector/ODBC Community Support”](#).

Many problems can be resolved by upgrading your Connector/ODBC drivers to the latest available release. On Windows, you should also make sure that you have the latest versions of the Microsoft Data Access Components (MDAC) installed.

Questions

- [27.1.6.3.1](#): Are MyODBC 2.50 applications compatible with Connector/ODBC 3.51?
- [27.1.6.3.2](#): I have installed Connector/ODBC on Windows XP x64 Edition or Windows Server 2003 R2 x64. The installation com-

pleted successfully, but the Connector/ODBC driver does not appear in [ODBC Data Source Administrator](#).

- [27.1.6.3.3](#): When connecting or using the TEST button in [ODBC Data Source Administrator](#) I get error 10061 (Cannot connect to server)
- [27.1.6.3.4](#): The following error is reported when using transactions: [Transactions are not enabled](#)
- [27.1.6.3.5](#): The following error is reported when I submit a query: [Cursor not found](#)
- [27.1.6.3.6](#): Access reports records as [#DELETED#](#) when inserting or updating records in linked tables.
- [27.1.6.3.7](#): How do I handle Write Conflicts or Row Location errors?
- [27.1.6.3.8](#): Exporting data from Access 97 to MySQL reports a [Syntax Error](#).
- [27.1.6.3.9](#): Exporting data from Microsoft DTS to MySQL reports a [Syntax Error](#).
- [27.1.6.3.10](#): Using ODBC.NET with Connector/ODBC, while fetching empty string (0 length), it starts giving the SQL_NO_DATA exception.
- [27.1.6.3.11](#): Using [SELECT COUNT\(*\) FROM tbl_name](#) within Visual Basic and ASP returns an error.
- [27.1.6.3.12](#): Using the [AppendChunk\(\)](#) or [GetChunk\(\)](#) ADO methods, the [Multiple-step operation generated errors. Check each status value](#) error is returned.
- [27.1.6.3.13](#): Access Returns [Another user had modified the record that you have modified](#) while editing records on a Linked Table.
- [27.1.6.3.14](#): When linking an application directly to the Connector/ODBC library under Unix/Linux, the application crashes.
- [27.1.6.3.15](#): Applications in the Microsoft Office suite are unable to update tables that have [DATE](#) or [TIMESTAMP](#) columns.
- [27.1.6.3.16](#): When connecting Connector/ODBC 5.x (Beta) to a MySQL 4.x server, the error [1044 Access denied for user 'xxx'@'%' to database 'information_schema'](#) is returned.
- [27.1.6.3.17](#): When calling [SQLTables](#), the error [S1T00](#) is returned, but I cannot find this in the list of error numbers for Connector/ODBC.
- [27.1.6.3.18](#): When linking to tables in Access 2000 and generating links to tables programmatically, rather than through the table designer interface, you may get errors about tables not existing.
- [27.1.6.3.19](#): When I try to use batched statements, the execution of the batched statements fails.

Questions and Answers

27.1.6.3.1: Are MyODBC 2.50 applications compatible with Connector/ODBC 3.51?

Applications based on MyODBC 2.50 should work fine with Connector/ODBC 3.51 and later versions. If you find something is not working with the latest version of Connector/ODBC which previously worked under an earlier version, please file a bug report. See [Section 26.1.7.2, “How to Report Connector/ODBC Problems or Bugs”](#).

27.1.6.3.2: I have installed Connector/ODBC on Windows XP x64 Edition or Windows Server 2003 R2 x64. The installation completed successfully, but the Connector/ODBC driver does not appear in [ODBC Data Source Administrator](#).

This is not a bug, but is related to the way Windows x64 editions operate with the ODBC driver. On Windows x64 editions, the Connector/ODBC driver is installed in the [%SystemRoot%\SysWOW64](#) folder. However, the default [ODBC Data Source Administrator](#) that is available through the [Administrative Tools](#) or [Control Panel](#) in Windows x64 Editions is located in the [%SystemRoot%\system32](#) folder, and only searches this folder for ODBC drivers.

On Windows x64 editions, you should use the ODBC administration tool located at [%SystemRoot%\SysWOW64\odbcad32.exe](#), this will correctly locate the installed Connector/ODBC drivers and enable you to create a Connector/ODBC DSN.

This issue was originally reported as [Bug#20301](#).

27.1.6.3.3: When connecting or using the TEST button in [ODBC Data Source Administrator](#) I get error 10061 (Cannot

connect to server)

This error can be raised by a number of different issues, including server problems, network problems, and firewall and port blocking problems. For more information, see [Section B.1.2.2, “Can't connect to \[local\] MySQL server”](#).

27.1.6.3.4: The following error is reported when using transactions: Transactions are not enabled

This error indicates that you are trying to use transactions with a MySQL table that does not support transactions. Transactions are supported within MySQL when using the [InnoDB](#) database engine. In versions of MySQL before Mysql 5.1 you may also use the [BDB](#) engine.

You should check the following before continuing:

- Verify that your MySQL server supports a transactional database engine. Use [SHOW ENGINES](#) to obtain a list of the available engine types.
- Verify that the tables you are updating use a transaction database engine.
- Ensure that you have not enabled the [disable transactions](#) option in your DSN.

27.1.6.3.5: The following error is reported when I submit a query: Cursor not found

This occurs because the application is using the old MyODBC 2.50 version, and it did not set the cursor name explicitly through SQL-SetCursorName. The fix is to upgrade to Connector/ODBC 3.51 version.

27.1.6.3.6: Access reports records as #DELETED# when inserting or updating records in linked tables.

If the inserted or updated records are shown as [#DELETED#](#) in the access, then:

- If you are using Access 2000, you should get and install the newest (version 2.6 or higher) Microsoft MDAC ([Microsoft Data Access Components](#)) from <http://www.microsoft.com/data/>. This fixes a bug in Access that when you export data to MySQL, the table and column names aren't specified. Another way to work around this bug is to upgrade to MyODBC 2.50.33 or higher and MySQL 3.23.x or higher, which together provide a workaround for the problem.

You should also get and apply the Microsoft Jet 4.0 Service Pack 5 (SP5) which can be found at <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114>. This fixes some cases where columns are marked as [#DELETED#](#) in Access.

Note

If you are using MySQL 3.22, you must apply the MDAC patch and use MyODBC 2.50.32 or 2.50.34 and up to work around this problem.

- For all versions of Access, you should enable the Connector/ODBC [Return matching rows](#) option. For Access 2.0, you should additionally enable the [Simulate ODBC 1.0](#) option.
- You should have a timestamp in all tables that you want to be able to update.
- You should have a primary key in the table. If not, new or updated rows may show up as [#DELETED#](#).
- Use only [DOUBLE](#) float fields. Access fails when comparing with single-precision floats. The symptom usually is that new or updated rows may show up as [#DELETED#](#) or that you can't find or update rows.
- If you are using Connector/ODBC to link to a table that has a [BIGINT](#) column, the results are displayed as [#DELETED](#). The work around solution is:
 - Have one more dummy column with [TIMESTAMP](#) as the data type.
 - Select the [Change BIGINT columns to INT](#) option in the connection dialog in ODBC DSN Administrator.
 - Delete the table link from Access and re-create it.

Old records still display as [#DELETED#](#), but newly added/updated records are displayed properly.

27.1.6.3.7: How do I handle Write Conflicts or Row Location errors?

If you see the following errors, select the **Return Matching Rows** option in the DSN configuration dialog, or specify **OPTION=2**, as the connection parameter:

```
Write Conflict. Another user has changed your data.  
Row cannot be located for updating. Some values may have been changed  
since it was last read.
```

27.1.6.3.8: Exporting data from Access 97 to MySQL reports a Syntax Error.

This error is specific to Access 97 and versions of Connector/ODBC earlier than 3.51.02. Update to the latest version of the Connector/ODBC driver to resolve this problem.

27.1.6.3.9: Exporting data from Microsoft DTS to MySQL reports a Syntax Error.

This error occurs only with MySQL tables using the **TEXT** or **VARCHAR** data types. You can fix this error by upgrading your Connector/ODBC driver to version 3.51.02 or higher.

27.1.6.3.10: Using ODBC.NET with Connector/ODBC, while fetching empty string (0 length), it starts giving the SQL_NO_DATA exception.

You can get the patch that addresses this problem from <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q319243>.

27.1.6.3.11: Using `SELECT COUNT(*) FROM tbl_name` within Visual Basic and ASP returns an error.

This error occurs because the `COUNT(*)` expression is returning a **BIGINT**, and ADO can't make sense of a number this big. Select the **Change BIGINT columns to INT** option (option value 16384).

27.1.6.3.12: Using the `AppendChunk()` or `GetChunk()` ADO methods, the Multiple-step operation generated errors. Check each status value error is returned.

The `GetChunk()` and `AppendChunk()` methods from ADO doesn't work as expected when the cursor location is specified as **adUseServer**. On the other hand, you can overcome this error by using **adUseClient**.

A simple example can be found from http://www.dwam.net/iishelp/ado/docs/adomth02_4.htm

27.1.6.3.13: Access Returns `Another user had modified the record that you have modified` while editing records on a Linked Table.

In most cases, this can be solved by doing one of the following things:

- Add a primary key for the table if one doesn't exist.
- Add a timestamp column if one doesn't exist.
- Only use double-precision float fields. Some programs may fail when they compare single-precision floats.

If these strategies don't help, you should start by making a log file from the ODBC manager (the log you get when requesting logs from ODBCADMIN) and a Connector/ODBC log to help you figure out why things go wrong. For instructions, see [Section 26.1.3.8](#), "Getting an ODBC Trace File".

27.1.6.3.14: When linking an application directly to the Connector/ODBC library under Unix/Linux, the application crashes.

Connector/ODBC 3.51 under Unix/Linux is not compatible with direct application linking. You must use a driver manager, such as iODBC or unixODBC to connect to an ODBC source.

27.1.6.3.15: Applications in the Microsoft Office suite are unable to update tables that have **DATE or **TIMESTAMP** columns.**

This is a known issue with Connector/ODBC. You must ensure that the field has a default value (rather than **NULL** and that the default value is non-zero (i.e. the default value is not `0000-00-00 00:00:00`).

27.1.6.3.16: When connecting Connector/ODBC 5.x (Beta) to a MySQL 4.x server, the error `1044 Access denied for user 'xxx'@'%' to database 'information_schema'` is returned.

Connector/ODBC 5.x is designed to work with MySQL 5.0 or later, taking advantage of the [INFORMATION_SCHEMA](#) database to determine data definition information. Support for MySQL 4.1 is planned for the final release.

27.1.6.3.17: When calling [SQLTables](#), the error [S1T00](#) is returned, but I cannot find this in the list of error numbers for Connector/ODBC.

The [S1T00](#) error indicates that a general timeout has occurred within the ODBC system and is not a MySQL error. Typically it indicates that the connection you are using is stale, the server is too busy to accept your request or that the server has gone away.

27.1.6.3.18: When linking to tables in Access 2000 and generating links to tables programmatically, rather than through the table designer interface, you may get errors about tables not existing.

There is a known issue with a specific version of the [msjet40.dll](#) that exhibits this issue. The version affected is 4.0.9025.0. Reverting to an older version will enable you to create the links. If you have recently updated your version, check your [WINDOWS](#) directory for the older version of the file and copy it to the drivers directory.

27.1.6.3.19: When I try to use batched statements, the execution of the batched statements fails.

Batched statement support was added in 3.51.18. Support for batched statements is not enabled by default. You must enable option [FLAG_MULTI_STATEMENTS](#), value 67108864, or select the **ALLOW MULTIPLE STATEMENTS** flag within a GUI configuration.

26.1.7. Connector/ODBC Support

There are many different places where you can get support for using Connector/ODBC. You should always try the Connector/ODBC Mailing List or Connector/ODBC Forum. See [Section 26.1.7.1, “Connector/ODBC Community Support”](#), for help before reporting a specific bug or issue to MySQL.

26.1.7.1. Connector/ODBC Community Support

MySQL AB provides assistance to the user community by means of its mailing lists. For Connector/ODBC-related issues, you can get help from experienced users by using the [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com) mailing list. Archives are available online at <http://lists.mysql.com/myodbc>.

For information about subscribing to MySQL mailing lists or to browse list archives, visit <http://lists.mysql.com/>. See [Section 1.7.1, “MySQL Mailing Lists”](#).

Community support from experienced users is also available through the [ODBC Forum](#). You may also find help from other users in the other MySQL Forums, located at <http://forums.mysql.com>. See [Section 1.7.2, “MySQL Community Support at the MySQL Forums”](#).

26.1.7.2. How to Report Connector/ODBC Problems or Bugs

If you encounter difficulties or problems with Connector/ODBC, you should start by making a log file from the [ODBC Manager](#) (the log you get when requesting logs from [ODBC ADMIN](#)) and Connector/ODBC. The procedure for doing this is described in [Section 26.1.3.8, “Getting an ODBC Trace File”](#).

Check the Connector/ODBC trace file to find out what could be wrong. You should be able to determine what statements were issued by searching for the string [>mysql_real_query](#) in the [myodbc.log](#) file.

You should also try issuing the statements from the [mysql](#) client program or from [admdemo](#). This helps you determine whether the error is in Connector/ODBC or MySQL.

If you find out something is wrong, please only send the relevant rows (maximum 40 rows) to the [myodbc](#) mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#). Please never send the whole Connector/ODBC or ODBC log file!

You should ideally include the following information with the email:

- Operating system and version
- Connector/ODBC version
- ODBC Driver Manager type and version
- MySQL server version

- ODBC trace from Driver Manager
- Connector/ODBC log file from Connector/ODBC driver
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem!

Also, before posting the bug, check the MyODBC mailing list archive at <http://lists.mysql.com/myodbc>.

If you are unable to find out what's wrong, the last option is to create an archive in [tar](#) or Zip format that contains a Connector/ODBC trace file, the ODBC log file, and a [README](#) file that explains the problem. You can send this to <ftp://ftp.mysql.com/pub/mysql/upload/>. Only MySQL engineers have access to the files you upload, and we are very discreet with the data.

If you can create a program that also demonstrates the problem, please include it in the archive as well.

If the program works with another SQL server, you should include an ODBC log file where you perform exactly the same SQL statements so that we can compare the results between the two systems.

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

26.1.7.3. How to Submit a Connector/ODBC Patch

You can send a patch or suggest a better solution for any existing code or problems by sending a mail message to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com).

26.1.7.4. Connector/ODBC Change History

The Connector/ODBC Change History (Changelog) is located with the main Changelog for MySQL. See [Section C.2, “MySQL Connector/ODBC \(MyODBC\) Change History”](#).

26.1.7.5. Credits

These are the developers that have worked on the Connector/ODBC and Connector/ODBC 3.51 Drivers from MySQL AB.

- Michael (Monty) Widenius
- Venu Anuganti
- Peter Harvey

26.2. MySQL Connector/NET

Connector/NET enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. Connector/NET is a fully managed ADO.NET driver written in 100% pure C#.

Connector/NET includes full support for:

- MySQL 5.0 features (such as stored procedures)
- MySQL 4.1 features (server-side prepared statements, Unicode, and shared memory access, and so forth)
- Large-packet support for sending and receiving rows and BLOBs up to 2 gigabytes in size.
- Protocol compression which allows for compressing the data stream between the client and server.
- Support for connecting using TCP/IP sockets, named pipes, or shared memory on Windows.
- Support for connecting using TCP/IP sockets or Unix sockets on Unix.

- Support for the Open Source Mono framework developed by Novell.
- Fully managed, does not utilize the MySQL client library.

This document is intended as a user's guide to Connector/NET and includes a full syntax reference. Syntax information is also included within the [Documentation.chm](#) file included with the Connector/NET distribution.

If you are using MySQL 5.0 or later, and Visual Studio as your development environment, you may want also want to use the MySQL Visual Studio Plugin. The plugin acts as a DDEX (Data Designer Extensibility) provider, enabling you to use the data design tools within Visual Studio to manipulate the schema and objects within a MySQL database. For more information, see [Section 26.3, “MySQL Visual Studio Plugin”](#).

Note

Connector/NET 5.1.2 and later include the Visual Studio Plugin by default.

26.2.1. Connector/NET Versions

There are currently three versions of Connector/NET available:

- Connector/NET 1.0 includes support for MySQL 4.0, and MySQL 5.0 features, and full compatibility with the ADO.NET driver interface.

Connector/NET 5.0 includes support for MySQL 4.0, MySQL 4.1, MySQL 5.0 and MySQL 5.1 features. Connector/NET 5.0 also includes full support for the ADO.Net 2.0 interfaces and subclasses, includes support for the usage advisor and performance monitor (PerfMon) hooks.

Connector/NET 5.1 includes support for MySQL 4.0, MySQL 5.0, MySQL 5.1 and MySQL 6.0 (Falcon Preview) features. Connector/NET 5.1 also includes support for a new membership/role provider, Compact Framework 2.0, a new stored procedure parser and improvements to [GetSchema](#). Connector/NET 5.1 also includes the Visual Studio Plugin as a standard installable component.

Note

Version numbers for MySQL products are formatted as X.X.X. However, Windows tools (Control Panel, properties display) may show the version numbers as XX.XX.XX. For example, the official MySQL formatted version number 5.0.9 may be displayed by Windows tools as 5.00.09. The two versions are the same; only the number display format is different.

26.2.2. Connector/NET Installation

Connector/NET runs on any platform that supports the .NET framework. The .NET framework is primarily supported on recent versions of Microsoft Windows, and is supported on Linux through the Open Source Mono framework (see <http://www.mono-project.com>).

Connector/NET is available for download from <http://dev.mysql.com/downloads/connector/net/1.0.html>.

26.2.2.1. Installing Connector/NET on Windows

On Windows, installation is supported either through a binary installation process or by downloading a Zip file with the Connector/NET components.

Before installing, you should ensure that your system is up to date, including installing the latest version of the .NET Framework.

26.2.2.1.1. Installing Connector/NET using the Installer

Using the installer is the most straightforward method of installing Connector/NET on Windows and the installed components include the source code, test code and full reference documentation.

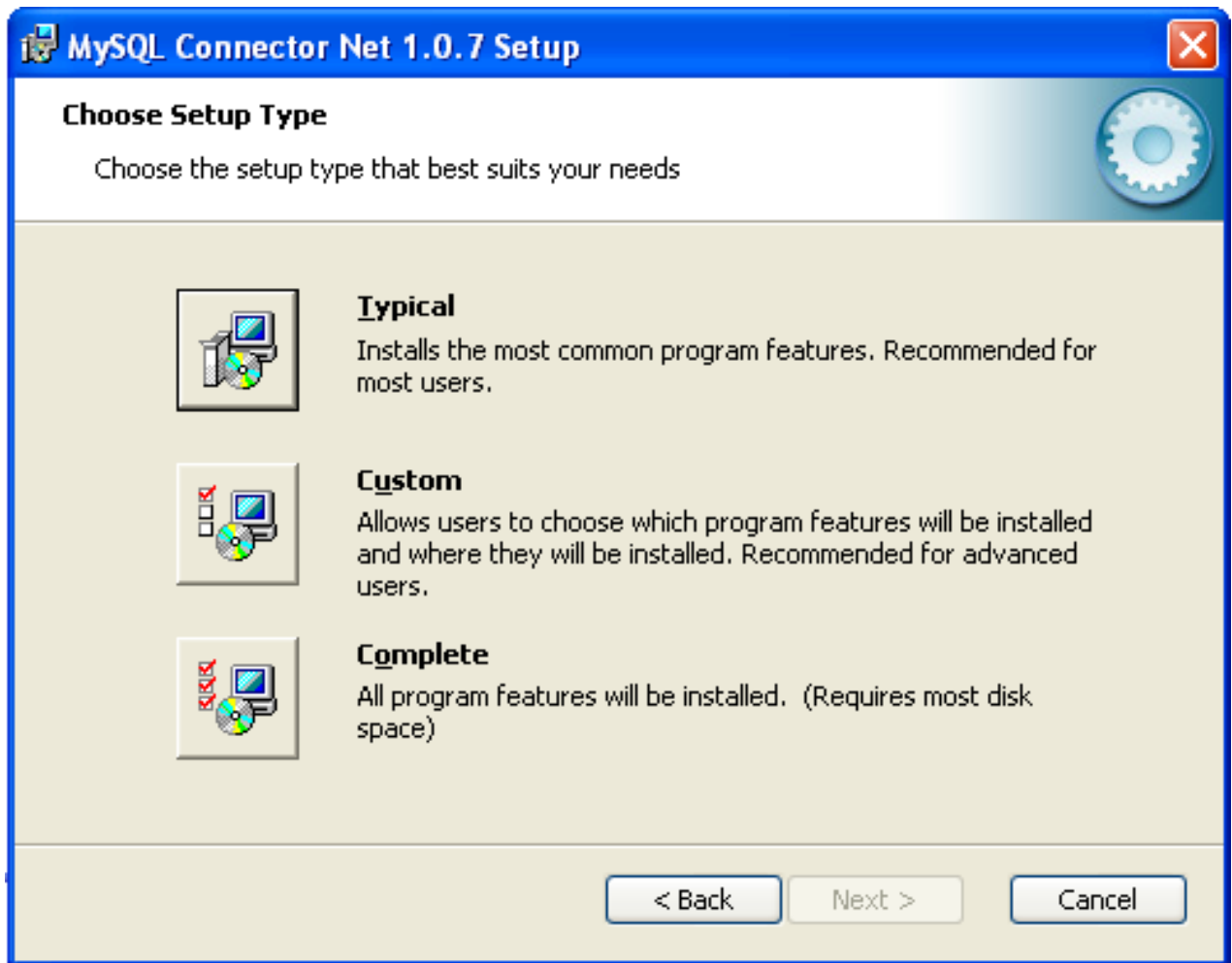
Connector/NET is installed through the use of a Windows Installer ([.msi](#)) installation package, which can be used to install Connector/NET on all Windows operating systems. The MSI package is contained within a ZIP archive named [mysql-connector-net-version.zip](#), where [version](#) indicates the Connector/NET version.

To install Connector/NET:

1. Double click on the MSI installer file extracted from the Zip you downloaded. Click NEXT to start the installation.



2. You must choose the type of installation that you want to perform.



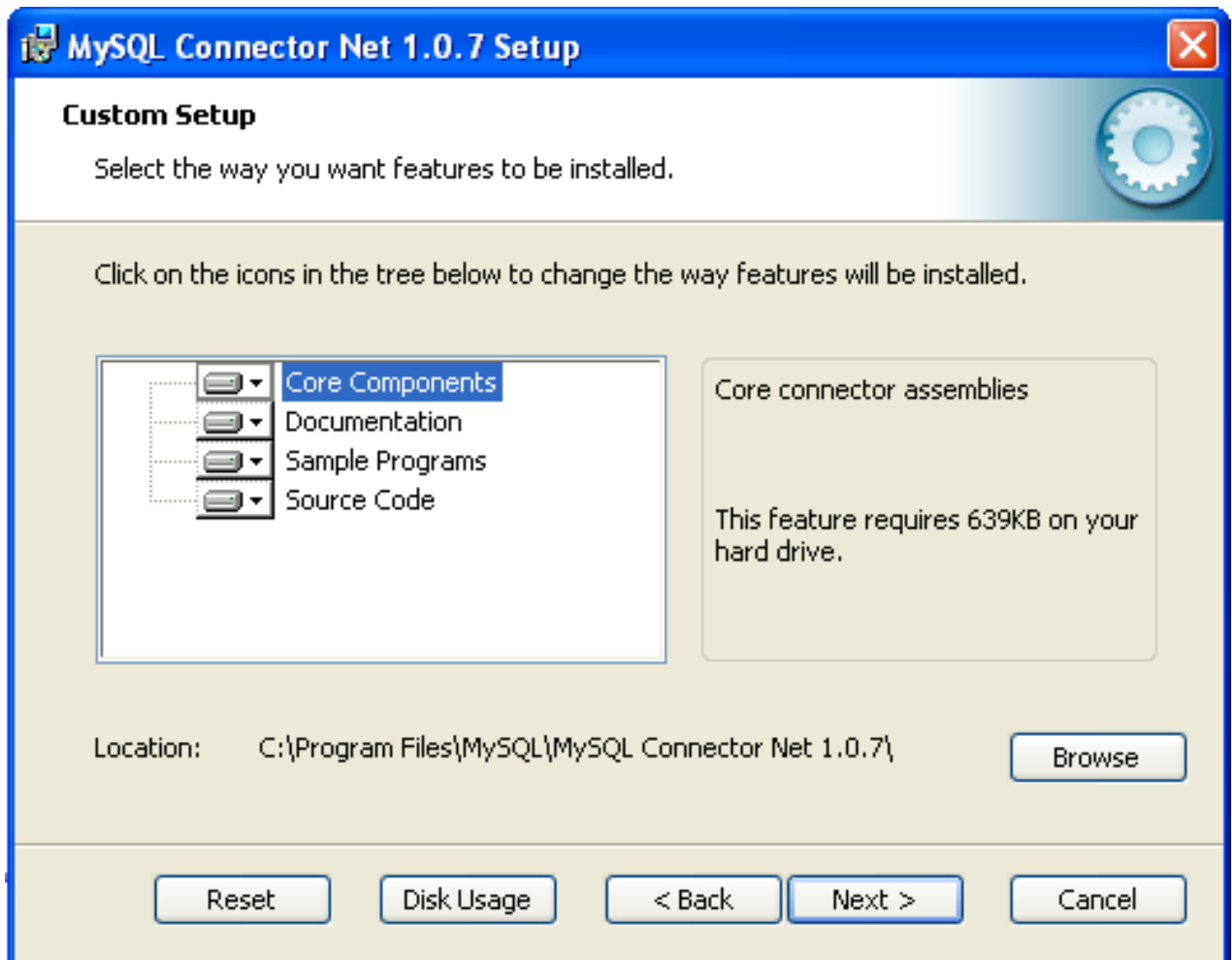
For most situations, the Typical installation will be suitable. Click the TYPICAL button and proceed to Step 5. A Complete installation installs all the available files. To conduct a Complete installation, click the COMPLETE button and proceed to step 5. If you want to customize your installation, including choosing the components to install and some installation options, click the CUSTOM button and proceed to Step 3.

The Connector/NET installer will register the connector within the Global Assembly Cache (GAC) - this will make the Connector/NET component available to all applications, not just those where you explicitly reference the Connector/NET component. The installer will also create the necessary links in the Start menu to the documentation and release notes.

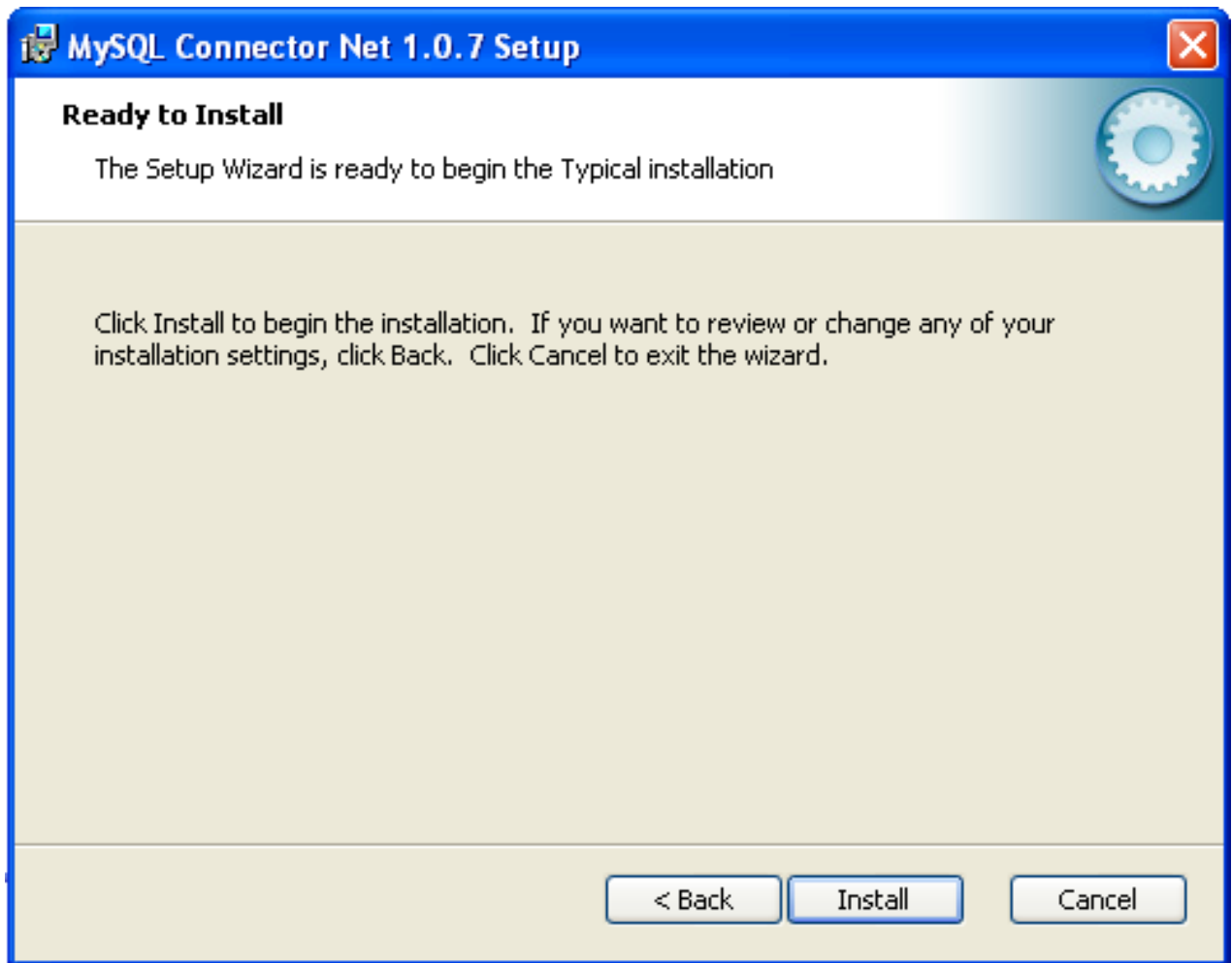
3. If you have chosen a custom installation, you can select the individual components that you want to install, including the core interface component, supporting documentation (a CHM file) samples and examples and the source code. Select the items, and their installation level, and then click NEXT to continue the installation.

Note

For Connector/NET 1.0.8 or lower and Connector 5.0.4 and lower the installer will attempt to install binaries for both 1.x and 2.x of the .NET Framework. If you only have one version of the framework installed, the connector installation may fail. If this happens, you can choose the framework version to be installed through the custom installation step.



4. You will be given a final opportunity to confirm the installation. Click INSTALL to copy and install the files onto your machine.



5. Once the installation has been completed, click FINISH to exit the installer.

Unless you choose otherwise, Connector/NET is installed in `C:\Program Files\MySQL\MySQL Connector Net X.X.X`, where `X.X.X` is replaced with the version of Connector/NET you are installing. New installations do not overwrite existing versions of Connector/NET.

Depending on your installation type, the installed components will include some or all of the following components:

- `bin` - Connector/NET MySQL libraries for different versions of the .NET environment.
- `docs` - contains a CHM of the Connector/NET documentation.
- `samples` - sample code and applications that use the Connector/NET component.
- `src` - the source code for the Connector/NET component.

You may also use the `/quiet` or `/q` command line option with the `msiexec` tool to install the Connector/NET package automatically (using the default options) with no notification to the user. Using this option you cannot select options and no prompts, messages or dialog boxes will be displayed.

```
C:\> msiexec /package conector-net.msi /quiet
```

To provide a progress bar to the user during automatic installation, but still without presenting the user with a dialog box of the ability

to select options, use the `/passive` option.

26.2.2.1.2. Installing Connector/NET using the Zip package

If you are having problems running the installer, you can download a .zip file without an installer as an alternative. That file is called `mysql-connector-net-version-noinstall.zip`. Once downloaded, you can extract the files to a location of your choice.

The .zip file contains the following directories:

- `bin` - Connector/NET MySQL libraries for different versions of the .NET environment.
- `doc` - contains a CHM of the Connector/NET documentation.
- `Samples` - sample code and applications that use the Connector/NET component.
- `mysqlclient` - the source code for the Connector/NET component.
- `testsuite` - the test suite used to verify the operation of the Connector/NET component.

26.2.2.2. Installing Connector/NET on Unix with Mono

There is no installer available for installing the Connector/NET component on your Unix installation. However, the installation is very simple. Before installing, please ensure that you have a working Mono project installation.

Note that you should only install the Connector/NET component on Unix environments where you want to connect to a MySQL server through the Mono project. If you are deploying or developing on a different environment such as Java or Perl then you should use a more appropriate connectivity component. See [Chapter 26, Connectors](#), or [Chapter 25, APIs and Libraries](#), for more information.

To install Connector/NET on Unix/Mono:

1. Download the `mysql-connector-net-version-noinstall.zip` and extract the contents.
2. Copy the `MySQL.Data.dll` file to your Mono project installation folder.
3. You must register the Connector/NET component in the Global Assembly Cache using the `gacutil` command:

```
shell> gacutil /i MySQL.Data.dll
```

Once installed, applications that are compiled with the Connector/NET component need no further changes. However, you must ensure that when you compile your applications you include the Connector/NET component using the `-r:MySQLData.dll` command line option.

26.2.2.3. Installing Connector/NET using the Source

Caution

You should read this section only if you are interested in helping us test our new code. If you just want to get Connector/NET up and running on your system, you should use a standard release distribution.

To be able to access the Connector/NET source tree, you must have Subversion installed. Subversion is freely available from <http://subversion.tigris.org/>.

The most recent development source tree is available from our public Subversion trees at <http://dev.mysql.com/tech-resources/sources.html>.

To checkout out the Connector/NET sources, change to the directory where you want the copy of the Connector/NET tree to be stored, then use the following command:

```
shell> svn co  
http://svn.mysql.com/svnpublic/connector-net
```

A Visual Studio project is included in the source which you can use to build Connector/NET.

26.2.3. Connector/NET Examples and Usage Guide

Connector/NET comprises several classes that are used to connect to the database, execute queries and statements, and manage query results.

The following are the major classes of Connector/NET:

- **MySqlCommand**: Represents an SQL statement to execute against a MySQL database.
- **MySqlCommandBuilder**: Automatically generates single-table commands used to reconcile changes made to a DataSet with the associated MySQL database.
- **MySqlConnection**: Represents an open connection to a MySQL Server database.
- **MySqlDataAdapter**: Represents a set of data commands and a database connection that are used to fill a dataset and update a MySQL database.
- **MySqlDataReader**: Provides a means of reading a forward-only stream of rows from a MySQL database.
- **MySqlException**: The exception that is thrown when MySQL returns an error.
- **MySqlHelper**: Helper class that makes it easier to work with the provider.
- **MySqlTransaction**: Represents an SQL transaction to be made in a MySQL database.

This section contains basic information and examples for each of the above classes. For a more detailed reference guide please see [Section 26.2.4, “Connector/NET Reference”](#).

26.2.3.1. Using MySqlCommand

Represents a SQL statement to execute against a MySQL database. This class cannot be inherited.

MySqlCommand features the following methods for executing commands at a MySQL database:

Item	Description
ExecuteReader	Executes commands that return rows.
ExecuteNonQuery	Executes commands such as SQL INSERT, DELETE, and UPDATE statements.
ExecuteScalar	Retrieves a single value (for example, an aggregate value) from a database.

You can reset the **CommandText** property and reuse the **MySqlCommand** object. However, you must close the **MySqlDataReader** before you can execute a new or previous command.

If a **MySqlException** is generated by the method executing a **MySqlCommand**, the **MySqlConnection** remains open. It is the responsibility of the programmer to close the connection.

Note

Prior versions of the provider used the '@' symbol to mark parameters in SQL. This is incompatible with MySQL user variables, so the provider now uses the '?' symbol to locate parameters in SQL. To support older code, you can set 'old syntax=yes' on your connection string. If you do this, please be aware that an exception will not be thrown if you fail to define a parameter that you intended to use in your SQL.

Examples

The following example creates a **MySqlCommand** and a **MySqlConnection**. The **MySqlConnection** is opened and set as the **Connection** for the **MySqlCommand**. The example then calls **ExecuteNonQuery**, and closes the connection. To accomplish this, the

`ExecuteNonQuery` is passed a connection string and a query string that is a SQL INSERT statement.

Visual Basic example:

```
Public Sub InsertRow(myConnectionString As String)
    " If the connection string is null, use a default.
    If myConnectionString = "" Then
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass"
    End If
    Dim myConnection As New MySqlConnection(myConnectionString)
    Dim myInsertQuery As String = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)"
    Dim myCommand As New MySqlCommand(myInsertQuery)
    myCommand.Connection = myConnection
    myConnection.Open()
    myCommand.ExecuteNonQuery()
    myCommand.Connection.Close()
End Sub
```

C# example:

```
public void InsertRow(string myConnectionString)
{
    // If the connection string is null, use a default.
    if(myConnectionString == "")
    {
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass";
    }
    MySqlConnection myConnection = new MySqlConnection(myConnectionString);
    string myInsertQuery = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)";
    MySqlCommand myCommand = new MySqlCommand(myInsertQuery);
    myCommand.Connection = myConnection;
    myConnection.Open();
    myCommand.ExecuteNonQuery();
    myCommand.Connection.Close();
}
```

26.2.3.1.1. Class MySqlCommand Constructor Form 1

Overload methods for MySqlCommand

Initializes a new instance of the MySqlCommand class.

Examples

The following example creates a MySqlCommand and sets some of its properties.

Note

This example shows how to use one of the overloaded versions of the `MySqlCommand` constructor. For other examples that might be available, see the individual overload topics.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
    Dim myConnection As New MySqlConnection _
        ("Persist Security Info=False;database=test;server=myServer")
    myConnection.Open()
    Dim myTrans As MySqlTransaction = myConnection.BeginTransaction()
    Dim mySelectQuery As String = "SELECT * FROM MyTable"
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection, myTrans)
    myCommand.CommandTimeout = 20
End Sub
```

C# example:

```
public void CreateMySqlCommand()
{
    MySqlConnection myConnection = new MySqlConnection("Persist Security Info=False;
        database=test;server=myServer");
    myConnection.Open();
    MySqlTransaction myTrans = myConnection.BeginTransaction();
    string mySelectQuery = "SELECT * FROM myTable";
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnection, myTrans);
    myCommand.CommandTimeout = 20;
}
```

C++ example:

```
public:
void CreateMySQLCommand()
{
    MySqlConnection* myConnection = new MySqlConnection(S"Persist Security Info=False;
        database=test;server=myServer");
    myConnection->Open();
    MySqlTransaction* myTrans = myConnection->BeginTransaction();
    String* mySelectQuery = S"SELECT * FROM myTable";
    MySqlCommand* myCommand = new MySqlCommand(mySelectQuery, myConnection, myTrans);
    myCommand->CommandTimeout = 20;
};
```

Initializes a new instance of the `MySQLCommand` class.

The base constructor initializes all fields to their default values. The following table shows initial property values for an instance of `MySQLCommand`.

Properties	Initial Value
CommandText	empty string ("")
CommandTimeout	0
CommandType	<code>CommandType.Text</code>
Connection	Null

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a `MySQLCommand` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySQLCommand()
    Dim myCommand As New MySqlCommand()
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySQLCommand()
{
    MySqlCommand myCommand = new MySqlCommand();
    myCommand.CommandType = CommandType.Text;
}
```

26.2.3.1.2. Class `MySQLCommand` Constructor Form 2

Initializes a new instance of the `MySQLCommand` class with the text of the query.

Parameters: The text of the query.

When an instance of `MySQLCommand` is created, the following read/write properties are set to initial values.

Properties	Initial Value
CommandText	<code>cmdText</code>
CommandTimeout	0
CommandType	<code>CommandType.Text</code>
Connection	Null

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a [MySQLCommand](#) and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySQLCommand()  
    Dim sql as String = "SELECT * FROM mytable"  
    Dim myCommand As New MySQLCommand(sql)  
    myCommand.CommandType = CommandType.Text  
End Sub
```

C# example:

```
public void CreateMySQLCommand()  
{  
    string sql = "SELECT * FROM mytable";  
    MySqlCommand myCommand = new MySqlCommand(sql);  
    myCommand.CommandType = CommandType.Text;  
}
```

26.2.3.1.3. Class MySqlCommand Constructor Form 3

Initializes a new instance of the [MySQLCommand](#) class with the text of the query and a [MySQLConnection](#).

Parameters: The text of the query.

Parameters: A [MySQLConnection](#) that represents the connection to an instance of SQL Server.

When an instance of [MySQLCommand](#) is created, the following read/write properties are set to initial values.

Properties	Initial Value
CommandText	cmdText
CommandTimeout	0
CommandType	CommandType.Text
Connection	connection

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a [MySQLCommand](#) and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySQLCommand()  
    Dim conn as new MySQLConnection("server=myServer")  
    Dim sql as String = "SELECT * FROM mytable"  
    Dim myCommand As New MySqlCommand(sql, conn)  
    myCommand.CommandType = CommandType.Text  
End Sub
```

C# example:

```
public void CreateMySQLCommand()  
{  
    MySqlConnection conn = new MySqlConnection("server=myserver")  
    string sql = "SELECT * FROM mytable";  
    MySqlCommand myCommand = new MySqlCommand(sql, conn);  
    myCommand.CommandType = CommandType.Text;  
}
```

26.2.3.1.4. Class MySqlCommand Constructor Form 4

Initializes a new instance of the [MySQLCommand](#) class with the text of the query, a [MySQLConnection](#), and the [MySQLTransaction](#).

Parameters: The text of the query.

Parameters: A [MySQLConnection](#) that represents the connection to an instance of SQL Server.

Parameters: The [MySQLTransaction](#) in which the [MySQLCommand](#) executes.

When an instance of [MySQLCommand](#) is created, the following read/write properties are set to initial values.

Properties	Initial Value
CommandText	cmdText
CommandTimeout	0
CommandType	CommandType.Text
Connection	connection

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a [MySQLCommand](#) and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySQLCommand()
    Dim conn as new MySqlConnection("server=myServer")
    conn.Open();
    Dim txn as MySQLTransaction = conn.BeginTransaction()
    Dim sql as String = "SELECT * FROM mytable"
    Dim myCommand As New MySQLCommand(sql, conn, txn)
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySQLCommand()
{
    MySqlConnection conn = new MySqlConnection("server=myserver")
    conn.Open();
    MySQLTransaction txn = conn.BeginTransaction();
    string sql = "SELECT * FROM mytable";
    MySQLCommand myCommand = new MySQLCommand(sql, conn, txn);
    myCommand.CommandType = CommandType.Text;
}
```

26.2.3.1.5. ExecuteNonQuery

Executes a SQL statement against the connection and returns the number of rows affected.

Returns: Number of rows affected

You can use [ExecuteNonQuery](#) to perform any type of database operation, however any resultsets returned will not be available. Any output parameters used in calling a stored procedure will be populated with data and can be retrieved after execution is complete. For UPDATE, INSERT, and DELETE statements, the return value is the number of rows affected by the command. For all other types of statements, the return value is -1.

Examples

The following example creates a [MySQLCommand](#) and then executes it using [ExecuteNonQuery](#). The example is passed a string that is a SQL statement (such as UPDATE, INSERT, or DELETE) and a string to use to connect to the data source.

Visual Basic example:

```
Public Sub CreateMySQLCommand(myExecuteQuery As String, myConnection As MySqlConnection)
    Dim myCommand As New MySQLCommand(myExecuteQuery, myConnection)
    myCommand.Connection.Open()
```

```

myCommand.ExecuteNonQuery()
myConnection.Close()
End Sub

```

C# example:

```

public void CreateMySQLCommand(string myExecuteQuery, MySqlConnection myConnection)
{
    MySqlCommand myCommand = new MySqlCommand(myExecuteQuery, myConnection);
    myCommand.Connection.Open();
    myCommand.ExecuteNonQuery();
    myConnection.Close();
}

```

26.2.3.1.6. ExecuteReader1

Sends the [CommandText](#) to the [MySQLConnection](#)Connection, and builds a [MySQLDataReader](#) using one of the [CommandBehavior](#) values.

Parameters: One of the [CommandBehavior](#) values.

When the [CommandType](#) property is set to [StoredProcedure](#), the [CommandText](#) property should be set to the name of the stored procedure. The command executes this stored procedure when you call [ExecuteReader](#).

The [MySQLDataReader](#) supports a special mode that enables large binary values to be read efficiently. For more information, see the [SequentialAccess](#) setting for [CommandBehavior](#).

While the [MySQLDataReader](#) is in use, the associated [MySQLConnection](#) is busy serving the [MySQLDataReader](#). While in this state, no other operations can be performed on the [MySQLConnection](#) other than closing it. This is the case until the [MySQLDataReader.Close](#) method of the [MySQLDataReader](#) is called. If the [MySQLDataReader](#) is created with [CommandBehavior](#) set to [CloseConnection](#), closing the [MySQLDataReader](#) closes the connection automatically.

Note

When calling [ExecuteReader](#) with the [SingleRow](#) behavior, you should be aware that using a [limit](#) clause in your SQL will cause all rows (up to the limit given) to be retrieved by the client. The [MySQLDataReader.Read](#) method will still return false after the first row but pulling all rows of data into the client will have a performance impact. If the [limit](#) clause is not necessary, it should be avoided.

Returns: A [MySQLDataReader](#) object.

26.2.3.1.7. Using ExecuteReader

Sends the [CommandText](#) to the [MySQLConnection](#)Connection and builds a [MySQLDataReader](#).

Returns: A [MySQLDataReader](#) object.

When the [CommandType](#) property is set to [StoredProcedure](#), the [CommandText](#) property should be set to the name of the stored procedure. The command executes this stored procedure when you call [ExecuteReader](#).

While the [MySQLDataReader](#) is in use, the associated [MySQLConnection](#) is busy serving the [MySQLDataReader](#). While in this state, no other operations can be performed on the [MySQLConnection](#) other than closing it. This is the case until the [MySQLDataReader.Close](#) method of the [MySQLDataReader](#) is called.

Examples

The following example creates a [MySQLCommand](#), then executes it by passing a string that is a SQL [SELECT](#) statement, and a string to use to connect to the data source.

Visual Basic example:

```

Public Sub CreateMySQLDataReader(mySelectQuery As String, myConnection As MySqlConnection)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myConnection.Open()
    Dim myReader As MySQLDataReader
    myReader = myCommand.ExecuteReader()
    Try

```

```
While myReader.Read()  
    Console.WriteLine(myReader.GetString(0))  
End While  
Finally  
    myReader.Close  
    myConnection.Close  
End Try  
End Sub
```

C# example:

```
public void CreateMySqlDataReader(string mySelectQuery, MySqlConnection myConnection)  
{  
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnection);  
    myConnection.Open();  
    MySqlDataReader myReader;  
    myReader = myCommand.ExecuteReader();  
    try  
    {  
        while(myReader.Read())  
        {  
            Console.WriteLine(myReader.GetString(0));  
        }  
    }  
    finally  
    {  
        myReader.Close();  
        myConnection.Close();  
    }  
}
```

26.2.3.1.8. Using **Prepare**

Creates a prepared version of the command on an instance of MySQL Server.

Prepared statements are only supported on MySQL version 4.1 and higher. Calling prepare while connected to earlier versions of MySQL will succeed but will execute the statement in the same way as unprepared.

Examples

The following example demonstrates the use of the **Prepare** method.

Visual Basic example:

```
public sub PrepareExample()  
    Dim cmd as New MySqlCommand("INSERT INTO mytable VALUES (?val)", myConnection)  
    cmd.Parameters.Add( "?val", 10 )  
    cmd.Prepare()  
    cmd.ExecuteNonQuery()  
  
    cmd.Parameters(0).Value = 20  
    cmd.ExecuteNonQuery()  
end sub
```

C# example:

```
private void PrepareExample()  
{  
    MySqlCommand cmd = new MySqlCommand("INSERT INTO mytable VALUES (?val)", myConnection);  
    cmd.Parameters.Add( "?val", 10 );  
    cmd.Prepare();  
    cmd.ExecuteNonQuery();  
  
    cmd.Parameters[0].Value = 20;  
    cmd.ExecuteNonQuery();  
}
```

26.2.3.1.9. ExecuteScalar

Executes the query, and returns the first column of the first row in the result set returned by the query. Extra columns or rows are ignored.

Returns: The first column of the first row in the result set, or a null reference if the result set is empty

Use the [ExecuteScalar](#) method to retrieve a single value (for example, an aggregate value) from a database. This requires less code than using the [ExecuteReader](#) method, and then performing the operations necessary to generate the single value using the data returned by a [MySqlDataReader](#).

A typical [ExecuteScalar](#) query can be formatted as in the following C# example:

C# example:

```
cmd.CommandText = "select count(*) from region";
Int32 count = (int32) cmd.ExecuteScalar();
```

Examples

The following example creates a [MySqlCommand](#) and then executes it using [ExecuteScalar](#). The example is passed a string that is a SQL statement that returns an aggregate result, and a string to use to connect to the data source.

Visual Basic example:

```
Public Sub CreateMySqlCommand(myScalarQuery As String, myConnection As MySqlConnection)
    Dim myCommand As New MySqlCommand(myScalarQuery, myConnection)
    myCommand.Connection.Open()
    myCommand.ExecuteScalar()
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySqlCommand(string myScalarQuery, MySqlConnection myConnection)
{
    MySqlCommand myCommand = new MySqlCommand(myScalarQuery, myConnection);
    myCommand.Connection.Open();
    myCommand.ExecuteScalar();
    myConnection.Close();
}
```

C++ example:

```
public:
    void CreateMySqlCommand(String* myScalarQuery, MySqlConnection* myConnection)
    {
        MySqlCommand* myCommand = new MySqlCommand(myScalarQuery, myConnection);
        myCommand->Connection->Open();
        myCommand->ExecuteScalar();
        myConnection->Close();
    }
```

26.2.3.1.10. CommandText

Gets or sets the SQL statement to execute at the data source.

Value: The SQL statement or stored procedure to execute. The default is an empty string.

When the [CommandType](#) property is set to [StoredProcedure](#), the [CommandText](#) property should be set to the name of the stored procedure. The user may be required to use escape character syntax if the stored procedure name contains any special characters. The command executes this stored procedure when you call one of the Execute methods.

Examples

The following example creates a [MySqlCommand](#) and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
    Dim myCommand As New MySqlCommand()
    myCommand.CommandText = "SELECT * FROM Mytable ORDER BY id"
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySqlCommand()
{
    MySqlCommand myCommand = new MySqlCommand();
    myCommand.CommandText = "SELECT * FROM mytable ORDER BY id";
    myCommand.CommandType = CommandType.Text;
}
```

26.2.3.1.11. CommandTimeout

Gets or sets the wait time before terminating the attempt to execute a command and generating an error.

Value: The time (in seconds) to wait for the command to execute. The default is 0 seconds.

MySQL currently does not support any method of canceling a pending or executing operation. All commands issued against a MySQL server will execute until completion or until an exception occurs.

MySQL Enterprise

MySQL Enterprise subscribers will find more information about `CommandTimeout` in the Knowledge Base article, [Why CommandTimeout is not Supported](https://www.mysql.com/products/enterprise/advisors.html). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <https://www.mysql.com/products/enterprise/advisors.html>.

26.2.3.1.12. CommandType

Gets or sets a value indicating how the `CommandText` property is to be interpreted.

Value: One of the `System.Data.CommandType` values. The default is `Text`.

When you set the `CommandType` property to `StoredProcedure`, you should set the `CommandText` property to the name of the stored procedure. The command executes this stored procedure when you call one of the `Execute` methods.

Examples

The following example creates a `MySqlCommand` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
    Dim myCommand As New MySqlCommand()
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySqlCommand()
{
    MySqlCommand myCommand = new MySqlCommand();
    myCommand.CommandType = CommandType.Text;
}
```

26.2.3.1.13. Connection

Gets or sets the `MySqlConnection` used by this instance of the `MySqlCommand`.

Value: The connection to a data source. The default value is a null reference (`Nothing` in Visual Basic).

If you set `Connection` while a transaction is in progress and the `Transaction` property is not null, an `InvalidOperationException` is generated. If the `Transaction` property is not null and the transaction has already been committed or rolled back, `Transaction` is set to null.

Examples

The following example creates a `MySqlCommand` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()
    Dim mySelectQuery As String = "SELECT * FROM mytable ORDER BY id"
```

```

Dim myConnectionString As String = "Persist Security Info=False;database=test;server=myServer"
Dim myCommand As New MySqlCommand(mySelectQuery)
myCommand.Connection = New MySqlConnection(myConnectionString)
myCommand.CommandType = CommandType.Text
End Sub

```

C# example:

```

public void CreateMySqlCommand()
{
    string mySelectQuery = "SELECT * FROM mytable ORDER BY id";
    string myConnectionString = "Persist Security Info=False;database=test;server=myServer";
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery);
    myCommand.Connection = new MySqlConnection(myConnectionString);
    myCommand.CommandType = CommandType.Text;
}

```

26.2.3.1.14. IsPrepared

Returns true if the statement is prepared.

26.2.3.1.15. Parameters

Get the [MySqlParameterCollection](#)

Value: The parameters of the SQL statement or stored procedure. The default is an empty collection.

Connector/Net does not support unnamed parameters. Every parameter added to the collection must have an associated name.

Examples

The following example creates a [MySqlCommand](#) and displays its parameters. To accomplish this, the method is passed a [MySqlConnection](#), a query string that is a SQL [SELECT](#) statement, and an array of [MySqlParameter](#) objects.

Visual Basic example:

```

Public Sub CreateMySqlCommand(myConnection As MySqlConnection, _
mySelectQuery As String, myParamArray() As SqlParameter)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myCommand.CommandText = "SELECT id, name FROM mytable WHERE age=?age"
    myCommand.UpdatedRowSource = UpdateRowSource.Both
    myCommand.Parameters.Add(myParamArray)
    Dim j As Integer
    For j = 0 To myCommand.Parameters.Count - 1
        myCommand.Parameters.Add(myParamArray(j))
    Next j
    Dim myMessage As String = ""
    Dim i As Integer
    For i = 0 To myCommand.Parameters.Count - 1
        myMessage += myCommand.Parameters(i).ToString() & ControlChars.Cr
    Next i
    Console.WriteLine(myMessage)
End Sub

```

C# example:

```

public void CreateMySqlCommand(MySqlConnection myConnection, string mySelectQuery,
    SqlParameter[] myParamArray)
{
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnection);
    myCommand.CommandText = "SELECT id, name FROM mytable WHERE age=?age";
    myCommand.Parameters.Add(myParamArray);
    for (int j=0; j<myParamArray.Length; j++)
    {
        myCommand.Parameters.Add(myParamArray[j]) ;
    }
    string myMessage = "";
    for (int i = 0; i < myCommand.Parameters.Count; i++)
    {
        myMessage += myCommand.Parameters[i].ToString() + "\n";
    }
    MessageBox.Show(myMessage);
}

```

26.2.3.1.16. Transaction

Gets or sets the `MySqlCommandTransaction` within which the `MySqlCommand` executes.

Value: The `MySqlCommandTransaction`. The default value is a null reference (`Nothing` in Visual Basic).

You cannot set the `Transaction` property if it is already set to a specific value, and the command is in the process of executing. If you set the transaction property to a `MySqlCommandTransaction` object that is not connected to the same `MySqlConnection` as the `MySqlCommand` object, an exception will be thrown the next time you attempt to execute a statement.

26.2.3.1.17. UpdatedRowSource

Gets or sets how command results are applied to the `DataRow` when used by the `System.Data.Common.DbDataAdapter.Update` method of the `System.Data.Common.DbDataAdapter`.

Value: One of the `UpdateRowSource` values.

The default `System.Data.UpdateRowSource` value is `Both` unless the command is automatically generated (as in the case of the `MySqlCommandBuilder`), in which case the default is `None`.

26.2.3.2. Using MySqlCommandBuilder

Automatically generates single-table commands used to reconcile changes made to a `DataSet` with the associated MySQL database. This class cannot be inherited.

The `MySqlDataAdapter` does not automatically generate the SQL statements required to reconcile changes made to a `System.Data.DataSet` with the associated instance of MySQL. However, you can create a `MySqlCommandBuilder` object to automatically generate SQL statements for single-table updates if you set the `MySqlDataAdapter.SelectCommand` property of the `MySqlDataAdapter`. Then, any additional SQL statements that you do not set are generated by the `MySqlCommandBuilder`.

The `MySqlCommandBuilder` registers itself as a listener for `MySqlDataAdapter.OnRowUpdating` events whenever you set the `DataAdapter` property. You can only associate one `MySqlDataAdapter` or `MySqlCommandBuilder` object with each other at one time.

To generate INSERT, UPDATE, or DELETE statements, the `MySqlCommandBuilder` uses the `SelectCommand` property to retrieve a required set of metadata automatically. If you change the `SelectCommand` after the metadata has been retrieved (for example, after the first update), you should call the `RefreshSchema` method to update the metadata.

The `SelectCommand` must also return at least one primary key or unique column. If none are present, an `InvalidOperationException` exception is generated, and the commands are not generated.

When using `MySqlCommandBuilder` and `INSERT` you should set the `ReturnGeneratedIdentifiers` property to `true` to ensure that `AUTO_INCREMENT` fields in MySQL tables return the automatically generated value.

The `MySqlCommandBuilder` also uses the `MySqlCommand.Connection`, `MySqlCommand.CommandTimeout`, and `MySqlCommand.Transaction` properties referenced by the `SelectCommand`. The user should call `RefreshSchema` if any of these properties are modified, or if the `SelectCommand` itself is replaced. Otherwise the `MySqlDataAdapter.InsertCommand`, `MySqlDataAdapter.UpdateCommand`, and `MySqlDataAdapter.DeleteCommand` properties retain their previous values.

If you call `Dispose`, the `MySqlCommandBuilder` is disassociated from the `MySqlDataAdapter`, and the generated commands are no longer used.

Note

Caution must be used when using `MySqlCommandBuilder` on MySQL 4.0 systems. With MySQL 4.0, database/schema information is not provided to the connector for a query. This means that a query that pulls columns from two identically named tables in two or more different databases will not cause an exception to be thrown but will not work correctly. Even more dangerous is the situation where your select statement references database X but is executed in database Y and both databases have tables with similar layouts. This situation can cause unwanted changes or deletes. This note does not apply to MySQL versions 4.1 and later.

Examples

The following example uses the [MySqlCommand](#) , along [MySqlDataAdapter](#) and [MySqlConnection](#) , to select rows from a data source. The example is passed an initialized [System.Data.DataSet](#) , a connection string, a query string that is a SQL [SELECT](#) statement, and a string that is the name of the database table. The example then creates a [MySqlCommandBuilder](#) .

Visual Basic example:

```
Public Shared Function SelectRows(myConnection As String, mySelectQuery As String, myTableName As String) As DataSet
    Dim myConn As New MySqlConnection(myConnection)
    Dim myDataAdapter As New MySqlDataAdapter()
    myDataAdapter.SelectCommand = New MySqlCommand(mySelectQuery, myConn)
    Dim cb As SqlCommandBuilder = New MySqlCommandBuilder(myDataAdapter)
    myConn.Open()
    Dim ds As DataSet = New DataSet
    myDataAdapter.Fill(ds, myTableName)
    ' Code to modify data in DataSet here
    ' Without the MySqlCommandBuilder this line would fail.
    myDataAdapter.Update(ds, myTableName)
    myConn.Close()
End Function 'SelectRows
```

C# example:

```
public static DataSet SelectRows(string myConnection, string mySelectQuery, string myTableName)
{
    MySqlConnection myConn = new MySqlConnection(myConnection);
    MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();
    myDataAdapter.SelectCommand = new MySqlCommand(mySelectQuery, myConn);
    MySqlCommandBuilder cb = new MySqlCommandBuilder(myDataAdapter);
    myConn.Open();
    DataSet ds = new DataSet();
    myDataAdapter.Fill(ds, myTableName);
    //code to modify data in DataSet here
    //Without the MySqlCommandBuilder this line would fail
    myDataAdapter.Update(ds, myTableName);
    myConn.Close();
    return ds;
}
```

26.2.3.2.1. Class MySqlCommandBuilder Constructor

Initializes a new instance of the [MySqlCommandBuilder](#) class.

26.2.3.2.2. Class MySqlCommandBuilder Constructor Form 1

Initializes a new instance of the [MySqlCommandBuilder](#) class and sets the last one wins property.

Parameters: False to generate change protection code. True otherwise.

The [lastOneWins](#) parameter indicates whether SQL code should be included with the generated DELETE and UPDATE commands that checks the underlying data for changes. If [lastOneWins](#) is true then this code is not included and data records could be overwritten in a multi-user or multi-threaded environments. Setting [lastOneWins](#) to false will include this check which will cause a concurrency exception to be thrown if the underlying data record has changed without our knowledge.

26.2.3.2.3. Class MySqlCommandBuilder Constructor Form 2

Initializes a new instance of the [MySqlCommandBuilder](#) class with the associated [MySqlDataAdapter](#) object.

Parameters: The [MySqlDataAdapter](#) to use.

The [MySqlCommandBuilder](#) registers itself as a listener for [MySqlDataAdapter.RowUpdating](#) events that are generated by the [MySqlDataAdapter](#) specified in this property.

When you create a new instance [MySqlCommandBuilder](#) , any existing [MySqlCommandBuilder](#) associated with this [MySqlDataAdapter](#) is released.

26.2.3.2.4. Class MySqlCommandBuilder Constructor Form 3

Initializes a new instance of the [MySqlCommandBuilder](#) class with the associated [MySqlDataAdapter](#) object.

Parameters: The [MySqlDataAdapter](#) to use.

Parameters: False to generate change protection code. True otherwise.

The `MySQLCommandBuilder` registers itself as a listener for `MySQLDataAdapter.RowUpdating` events that are generated by the `MySQLDataAdapter` specified in this property.

When you create a new instance `MySQLCommandBuilder`, any existing `MySQLCommandBuilder` associated with this `MySQLDataAdapter` is released.

The `lastOneWins` parameter indicates whether SQL code should be included with the generated DELETE and UPDATE commands that checks the underlying data for changes. If `lastOneWins` is true then this code is not included and data records could be overwritten in a multi-user or multi-threaded environments. Setting `lastOneWins` to false will include this check which will cause a concurrency exception to be thrown if the underlying data record has changed without our knowledge.

26.2.3.2.5. DataAdapter

Gets or sets a `MySQLDataAdapter` object for which SQL statements are automatically generated.

Value: A `MySQLDataAdapter` object.

The `MySQLCommandBuilder` registers itself as a listener for `MySQLDataAdapter.RowUpdating` events that are generated by the `MySQLDataAdapter` specified in this property.

When you create a new instance `MySQLCommandBuilder`, any existing `MySQLCommandBuilder` associated with this `MySQLDataAdapter` is released.

26.2.3.2.6. QuotePrefix

Gets or sets the beginning character or characters to use when specifying MySQL database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.

Value: The beginning character or characters to use. The default value is `.

Database objects in MySQL can contain special characters such as spaces that would make normal SQL strings impossible to correctly parse. Use of the `QuotePrefix` and the `QuoteSuffix` properties allows the `MySQLCommandBuilder` to build SQL commands that handle this situation.

26.2.3.2.7. QuoteSuffix

Gets or sets the beginning character or characters to use when specifying MySQL database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.

Value: The beginning character or characters to use. The default value is `.

Database objects in MySQL can contain special characters such as spaces that would make normal SQL strings impossible to correctly parse. Use of the `QuotePrefix` and the `QuoteSuffix` properties allows the `MySQLCommandBuilder` to build SQL commands that handle this situation.

26.2.3.2.8. DeriveParameters

26.2.3.2.9. GetDeleteCommand

Gets the automatically generated `MySQLCommand` object required to perform deletions on the database.

Returns: The `MySQLCommand` object generated to handle delete operations.

An application can use the `GetDeleteCommand` method for informational or troubleshooting purposes because it returns the `MySQLCommand` object to be executed.

You can also use `GetDeleteCommand` as the basis of a modified command. For example, you might call `GetDeleteCommand` and modify the `MySQLCommand.CommandTimeout` value, and then explicitly set that on the `MySQLDataAdapter`.

After the SQL statement is first generated, the application must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetDeleteCommand` will be still be using information from the previous statement, which might not be correct.

The SQL statements are first generated either when the application calls `System.Data.Common.DataAdapter.Update` or `GetDeleteCommand`.

26.2.3.2.10. GetInsertCommand

Gets the automatically generated `MySqlCommand` object required to perform insertions on the database.

Returns: The `MySqlCommand` object generated to handle insert operations.

An application can use the `GetInsertCommand` method for informational or troubleshooting purposes because it returns the `MySqlCommand` object to be executed.

You can also use the `GetInsertCommand` as the basis of a modified command. For example, you might call `GetInsertCommand` and modify the `MySqlCommand.CommandTimeout` value, and then explicitly set that on the `MySqlDataAdapter`.

After the SQL statement is first generated, the application must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetInsertCommand` will be still be using information from the previous statement, which might not be correct. The SQL statements are first generated either when the application calls `System.Data.Common.DataAdapter.Update` or `GetInsertCommand`.

26.2.3.2.11. GetUpdateCommand

Gets the automatically generated `MySqlCommand` object required to perform updates on the database.

Returns: The `MySqlCommand` object generated to handle update operations.

An application can use the `GetUpdateCommand` method for informational or troubleshooting purposes because it returns the `MySqlCommand` object to be executed.

You can also use `GetUpdateCommand` as the basis of a modified command. For example, you might call `GetUpdateCommand` and modify the `MySqlCommand.CommandTimeout` value, and then explicitly set that on the `MySqlDataAdapter`.

After the SQL statement is first generated, the application must explicitly call `RefreshSchema` if it changes the statement in any way. Otherwise, the `GetUpdateCommand` will be still be using information from the previous statement, which might not be correct. The SQL statements are first generated either when the application calls `System.Data.Common.DataAdapter.Update` or `GetUpdateCommand`.

26.2.3.2.12. RefreshSchema

Refreshes the database schema information used to generate INSERT, UPDATE, or DELETE statements.

An application should call `RefreshSchema` whenever the `SELECT` statement associated with the `MySqlCommandBuilder` changes.

An application should call `RefreshSchema` whenever the `MySqlDataAdapter.SelectCommand` value of the `MySqlDataAdapter` changes.

MySQL Enterprise

MySQL Enterprise subscribers will find more information on this topic in the Knowledge Base article, [Understanding MySqlCommandBuilder and the LastOneWins Setting](http://www.mysql.com/products/enterprise/advisors.html). For information about subscribing to MySQL Enterprise see <http://www.mysql.com/products/enterprise/advisors.html>.

26.2.3.3. Using MySqlConnection

Represents an open connection to a MySQL Server database. This class cannot be inherited.

A `MySqlConnection` object represents a session to a MySQL Server data source. When you create an instance of `MySqlConnection`, all properties are set to their initial values. For a list of these values, see the `MySqlConnection` constructor.

If the `MySqlConnection` goes out of scope, it is not closed. Therefore, you must explicitly close the connection by calling `MySqlConnection.Close` or `MySqlConnection.Dispose`.

Examples

The following example creates a `MySqlCommand` and a `MySqlConnection`. The `MySqlConnection` is opened and set as the `MySqlCommand.Connection` for the `MySqlCommand`. The example then calls `MySqlCommand.ExecuteNonQuery`, and

closes the connection. To accomplish this, the `ExecuteNonQuery` is passed a connection string and a query string that is a SQL INSERT statement.

Visual Basic example:

```
Public Sub InsertRow(myConnectionString As String)
    ' If the connection string is null, use a default.
    If myConnectionString = "" Then
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass"
    End If
    Dim myConnection As New MySqlConnection(myConnectionString)
    Dim myInsertQuery As String = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)"
    Dim myCommand As New MySqlCommand(myInsertQuery)
    myCommand.Connection = myConnection
    myConnection.Open()
    myCommand.ExecuteNonQuery()
    myCommand.Connection.Close()
End Sub
```

C# example:

```
public void InsertRow(string myConnectionString)
{
    // If the connection string is null, use a default.
    if(myConnectionString == "")
    {
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass";
    }
    MySqlConnection myConnection = new MySqlConnection(myConnectionString);
    string myInsertQuery = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)";
    MySqlCommand myCommand = new MySqlCommand(myInsertQuery);
    myCommand.Connection = myConnection;
    myConnection.Open();
    myCommand.ExecuteNonQuery();
    myCommand.Connection.Close();
}
```

26.2.3.3.1. Class MySqlConnection Constructor (Default)

Initializes a new instance of the `MySqlConnection` class.

When a new instance of `MySqlConnection` is created, the read/write properties are set to the following initial values unless they are specifically set using their associated keywords in the `ConnectionString` property.

Properties	Initial Value
<code>ConnectionString</code>	empty string ("")
<code>ConnectionTimeout</code>	15
<code>Database</code>	empty string ("")
<code>DataSource</code>	empty string ("")
<code>ServerVersion</code>	empty string ("")

You can change the value for these properties only by using the `ConnectionString` property.

Examples

Overload methods for MySqlConnection

Initializes a new instance of the `MySqlConnection` class.

26.2.3.3.2. Class MySqlConnection Constructor Form 1

Initializes a new instance of the `MySqlConnection` class when given a string containing the connection string.

When a new instance of `MySqlConnection` is created, the read/write properties are set to the following initial values unless they are specifically set using their associated keywords in the `ConnectionString` property.

Properties	Initial Value

ConnectionString	empty string ("")
ConnectionTimeout	15
Database	empty string ("")
DataSource	empty string ("")
ServerVersion	empty string ("")

You can change the value for these properties only by using the [ConnectionString](#) property.

Examples

Parameters: The connection properties used to open the MySQL database.

26.2.3.3.3. Open

Opens a database connection with the property settings specified by the [ConnectionString](#).

Exception: Cannot open a connection without specifying a data source or server.

Exception: A connection-level error occurred while opening the connection.

The [MySQLConnection](#) draws an open connection from the connection pool if one is available. Otherwise, it establishes a new connection to an instance of MySQL.

Examples

The following example creates a [MySQLConnection](#), opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySQLConnection(myConnString As String)
    Dim myConnection As New MySQLConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
        + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySQLConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
        "\nState: " + myConnection.State.ToString());
    myConnection.Close();
}
```

26.2.3.3.4. Database

Gets the name of the current database or the database to be used after a connection is opened.

Returns: The name of the current database or the name of the database to be used after a connection is opened. The default value is an empty string.

The [Database](#) property does not update dynamically. If you change the current database using a SQL statement, then this property may reflect the wrong value. If you change the current database using the [ChangeDatabase](#) method, this property is updated to reflect the new database.

Examples

The following example creates a [MySQLConnection](#) and displays some of its read-only properties.

Visual Basic example:

```
Public Sub CreateMySQLConnection()  
    Dim myConnString As String = _  
        "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass"  
    Dim myConnection As New MySqlConnection( myConnString )  
    myConnection.Open()  
    MessageBox.Show( "Server Version: " + myConnection.ServerVersion _  
        + ControlChars.NewLine + "Database: " + myConnection.Database )  
    myConnection.ChangeDatabase( "test2" )  
    MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion _  
        + ControlChars.NewLine + "Database: " + myConnection.Database )  
    myConnection.Close()  
End Sub
```

C# example:

```
public void CreateMySQLConnection()  
{  
    string myConnString =  
        "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass";  
    MySqlConnection myConnection = new MySqlConnection( myConnString );  
    myConnection.Open();  
    MessageBox.Show( "Server Version: " + myConnection.ServerVersion  
        + "\nDatabase: " + myConnection.Database );  
    myConnection.ChangeDatabase( "test2" );  
    MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion  
        + "\nDatabase: " + myConnection.Database );  
    myConnection.Close();  
}
```

26.2.3.3.5. State

Gets the current state of the connection.

Returns: A bitwise combination of the [System.Data.ConnectionState](#) values. The default is [Closed](#).

The allowed state changes are:

- From [Closed](#) to [Open](#), using the [Open](#) method of the connection object.
- From [Open](#) to [Closed](#), using either the [Close](#) method or the [Dispose](#) method of the connection object.

Examples

The following example creates a [MySQLConnection](#), opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySQLConnection(myConnString As String)  
    Dim myConnection As New MySqlConnection(myConnString)  
    myConnection.Open()  
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _  
        + ControlChars.Cr + "State: " + myConnection.State.ToString())  
    myConnection.Close()  
End Sub
```

C# example:

```
public void CreateMySQLConnection(string myConnString)  
{  
    MySqlConnection myConnection = new MySqlConnection(myConnString);  
    myConnection.Open();  
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +  
        "\nState: " + myConnection.State.ToString());  
    myConnection.Close();  
}
```

26.2.3.3.6. ServerVersion

Gets a string containing the version of the MySQL server to which the client is connected.

Returns: The version of the instance of MySQL.

Exception: The connection is closed.

Examples

The following example creates a [MySQLConnection](#), opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySQLConnection(myConnString As String)
    Dim myConnection As New MySQLConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
        + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySQLConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
        "\nState: " + myConnection.State.ToString());
    myConnection.Close();
}
```

26.2.3.3.7. Close

Closes the connection to the database. This is the preferred method of closing any open connection.

The [Close](#) method rolls back any pending transactions. It then releases the connection to the connection pool, or closes the connection if connection pooling is disabled.

An application can call [Close](#) more than one time. No exception is generated.

Examples

The following example creates a [MySQLConnection](#), opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySQLConnection(myConnString As String)
    Dim myConnection As New MySQLConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
        + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySQLConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
        "\nState: " + myConnection.State.ToString());
    myConnection.Close();
}
```

26.2.3.3.8. CreateCommand

Creates and returns a [MySQLCommand](#) object associated with the [MySQLConnection](#).

Returns: A [MySQLCommand](#) object.

26.2.3.3.9. BeginTransaction

Begins a database transaction.

Returns: An object representing the new transaction.

Exception: Parallel transactions are not supported.

This command is equivalent to the MySQL BEGIN TRANSACTION command.

You must explicitly commit or roll back the transaction using the [MySQLTransaction.Commit](#) or [MySQLTransaction.Rollback](#) method.

Note

If you do not specify an isolation level, the default isolation level is used. To specify an isolation level with the [BeginTransaction](#) method, use the overload that takes the [iso](#) parameter.

Examples

The following example creates a [MySQLConnection](#) and a [MySQLTransaction](#). It also demonstrates how to use the [BeginTransaction](#), a [MySQLTransaction.Commit](#), and [MySQLTransaction.Rollback](#) methods.

Visual Basic example:

```
Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySQLTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into Test (id, desc) VALUES (100, 'Description')"
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into Test (id, desc) VALUES (101, 'Description')"
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Both records are written to database.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySqlException
            If Not myTrans.Connection Is Nothing Then
                Console.WriteLine("An exception of type " + ex.GetType().ToString() + _
                    " was encountered while attempting to roll back the transaction.")
            End If
        End Try

        Console.WriteLine("An exception of type " + e.GetType().ToString() + _
            " was encountered while inserting the data.")
        Console.WriteLine("Neither record was written to database.")
    Finally
        myConnection.Close()
    End Try
End Sub
```

C# example:

```
public void RunTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySQLTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
```

```

try
{
    myCommand.CommandText = "insert into Test (id, desc) VALUES (100, 'Description')";
    myCommand.ExecuteNonQuery();
    myCommand.CommandText = "insert into Test (id, desc) VALUES (101, 'Description')";
    myCommand.ExecuteNonQuery();
    myTrans.Commit();
    Console.WriteLine("Both records are written to database.");
}
catch(Exception e)
{
    try
    {
        myTrans.Rollback();
    }
    catch (SqlException ex)
    {
        if (myTrans.Connection != null)
        {
            Console.WriteLine("An exception of type " + ex.GetType() +
                " was encountered while attempting to roll back the transaction.");
        }
    }

    Console.WriteLine("An exception of type " + e.GetType() +
        " was encountered while inserting the data.");
    Console.WriteLine("Neither record was written to database.");
}
finally
{
    myConnection.Close();
}
}

```

26.2.3.3.10. BeginTransaction1

Begins a database transaction with the specified isolation level.

Parameters: The isolation level under which the transaction should run.

Returns: An object representing the new transaction.

Exception: Parallel exceptions are not supported.

This command is equivalent to the MySQL BEGIN TRANSACTION command.

You must explicitly commit or roll back the transaction using the [MySQLTransaction.Commit](#) or [MySQLTransaction.Rollback](#) method.

Note

If you do not specify an isolation level, the default isolation level is used. To specify an isolation level with the [BeginTransaction](#) method, use the overload that takes the [iso](#) parameter.

Examples

The following example creates a [MySQLConnection](#) and a [MySQLTransaction](#). It also demonstrates how to use the [BeginTransaction](#), a [MySQLTransaction.Commit](#), and [MySQLTransaction.Rollback](#) methods.

Visual Basic example:

```

Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySQLTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into Test (id, desc) VALUES (100, 'Description')"
        myCommand.ExecuteNonQuery()
    
```

```

        myCommand.CommandText = "Insert into Test (id, desc) VALUES (101, 'Description')"
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Both records are written to database.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySqlException
            If Not myTrans.Connection Is Nothing Then
                Console.WriteLine("An exception of type " + ex.GetType().ToString() + _
                    " was encountered while attempting to roll back the transaction.")
            End If
        End Try

        Console.WriteLine("An exception of type " + e.GetType().ToString() + _
            " was encountered while inserting the data.")
        Console.WriteLine("Neither record was written to database.")
    Finally
        myConnection.Close()
    End Try
End Sub

```

C# example:

```

public void RunTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "insert into Test (id, desc) VALUES (100, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "insert into Test (id, desc) VALUES (101, 'Description')";
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (SqlException ex)
        {
            if (myTrans.Connection != null)
            {
                Console.WriteLine("An exception of type " + ex.GetType() +
                    " was encountered while attempting to roll back the transaction.");
            }
        }

        Console.WriteLine("An exception of type " + e.GetType() +
            " was encountered while inserting the data.");
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
        myConnection.Close();
    }
}

```

26.2.3.3.11. ChangeDatabase

Changes the current database for an open MySqlConnection.

Parameters: The name of the database to use.

The value supplied in the [database](#) parameter must be a valid database name. The [database](#) parameter cannot contain a null value, an empty string, or a string with only blank characters.

When you are using connection pooling against MySQL, and you close the connection, it is returned to the connection pool. The next

time the connection is retrieved from the pool, the reset connection request executes before the user performs any operations.

MySQL Enterprise

MySQL Enterprise subscribers will find more information on this subject in the Knowledge Base article, [Understanding and Using Connection Pooling](#). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. To subscribe see <http://www.mysql.com/products/enterprise/advisors.html>.

Exception: The database name is not valid.

Exception: The connection is not open.

Exception: Cannot change the database.

Examples

The following example creates a `MySqlConnection` and displays some of its read-only properties.

Visual Basic example:

```
Public Sub CreateMySQLConnection()
    Dim myConnString As String = _
        "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass"
    Dim myConnection As New MySqlConnection( myConnString )
    myConnection.Open()
    MessageBox.Show( "Server Version: " + myConnection.ServerVersion _
        + ControlChars.NewLine + "Database: " + myConnection.Database )
    myConnection.ChangeDatabase( "test2" )
    MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion _
        + ControlChars.NewLine + "Database: " + myConnection.Database )
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySQLConnection()
{
    string myConnString =
        "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass";
    MySqlConnection myConnection = new MySqlConnection( myConnString );
    myConnection.Open();
    MessageBox.Show( "Server Version: " + myConnection.ServerVersion
        + "\nDatabase: " + myConnection.Database );
    myConnection.ChangeDatabase( "test2" );
    MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion
        + "\nDatabase: " + myConnection.Database );
    myConnection.Close();
}
```

26.2.3.3.12. StateChange

Occurs when the state of the connection changes.

The `StateChange` event fires whenever the `State` changes from closed to opened, or from opened to closed. `StateChange` fires immediately after the `MySqlConnection` transitions.

If an event handler throws an exception from within the `StateChange` event, the exception propagates to the caller of the `Open` or `Close` method.

The `StateChange` event is not raised unless you explicitly call `Close` or `Dispose`.

The event handler receives an argument of type `System.Data.StateChangeEventArgs` containing data related to this event. The following `StateChangeEventArgs` properties provide information specific to this event.

Property	Description
<code>System.Data.StateChangeEventArgs.CurrentState</code>	Gets the new state of the connection. The connection object will be in the new state already when the event is fired.
<code>System-</code>	Gets the original state of the connection.

<code>tem.Data.StateChangeEventArgs.OriginalState</code>	

26.2.3.3.13. InfoMessage

Occurs when MySQL returns warnings as a result of executing a command or query.

26.2.3.3.14. ConnectionTimeout

Gets the time to wait while trying to establish a connection before terminating the attempt and generating an error.

Exception: The value set is less than 0.

A value of 0 indicates no limit, and should be avoided in a `MySqlConnection.ConnectionString` because an attempt to connect will wait indefinitely.

Examples

The following example creates a `MySqlConnection` and sets some of its properties in the connection string.

Visual Basic example:

```
Public Sub CreateSqlConnection()
    Dim myConnection As New MySqlConnection()
    myConnection.ConnectionString = "Persist Security Info=False;Username=user;Password=pass;database=test1;server=localhost"
    myConnection.Open()
End Sub
```

C# example:

```
public void CreateSqlConnection()
{
    MySqlConnection myConnection = new MySqlConnection();
    myConnection.ConnectionString = "Persist Security Info=False;Username=user;»
        Password=pass;database=test1;server=localhost;Connect Timeout=30";
    myConnection.Open();
}
```

26.2.3.3.15. ConnectionString

Gets or sets the string used to connect to a MySQL Server database.

The `ConnectionString` returned may not be exactly like what was originally set but will be identical in terms of keyword/value pairs. Security information will not be included unless the `Persist Security Info` value is set to true.

You can use the `ConnectionString` property to connect to a database. The following example illustrates a typical connection string.

```
"Persist Security Info=False;database=MyDB;»
    server=MySqlServer;user id=myUser;Password=myPass"
```

The `ConnectionString` property can be set only when the connection is closed. Many of the connection string values have corresponding read-only properties. When the connection string is set, all of these properties are updated, except when an error is detected. In this case, none of the properties are updated. `MySqlConnection` properties return only those settings contained in the `ConnectionString`.

To connect to a local machine, specify "localhost" for the server. If you do not specify a server, localhost is assumed.

Resetting the `ConnectionString` on a closed connection resets all connection string values (and related properties) including the password. For example, if you set a connection string that includes "Database= MyDb", and then reset the connection string to "Data Source=myserver;User Id=myUser;Password=myPass", the `MySqlConnection.Database` property is no longer set to MyDb.

The connection string is parsed immediately after being set. If errors in syntax are found when parsing, a runtime exception, such as `ArgumentException`, is generated. Other errors can be found only when an attempt is made to open the connection.

The basic format of a connection string consists of a series of keyword/value pairs separated by semicolons. The equal sign (=) connects

each keyword and its value. Additional notes on setting values for options:

- To include values that contain a semicolon, single-quote character, or double-quote character, the value must be enclosed in double quotes. If the value contains both a semicolon and a double-quote character, the value can be enclosed in single quotes. The single quote is also useful if the value begins with a double-quote character. Conversely, the double quote can be used if the value begins with a single quote. If the value contains both single-quote and double-quote characters, the quote character used to enclose the value must be doubled each time it occurs within the value.
- To include preceding or trailing spaces in the string value, the value must be enclosed in either single quotes or double quotes. Any leading or trailing spaces around integer, Boolean, or enumerated values are ignored, even if enclosed in quotes. However, spaces within a string literal keyword or value are preserved. Using .NET Framework version 1.1, single or double quotes may be used within a connection string without using delimiters (for example, Data Source= my'Server or Data Source= my"Server), unless a quote character is the first or last character in the value.
- To include an equal sign (=) in a keyword or value, it must be preceded by another equal sign. For example, in the hypothetical connection string

```
"key==word=va lue"
```

the keyword is "key=word" and the value is "value".

- If a specific keyword in a keyword= value pair occurs multiple times in a connection string, the last occurrence listed is used in the value set.
- Keywords are not case sensitive.

The following table lists the valid names for keyword values within the [ConnectionString](#).

Name	Default	Description
Connect Timeout , Connection Timeout	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
Host , Server , Data Source , Data-Source , Address , Addr , Network Address	localhost	The name or network address of the instance of MySQL to which to connect. Multiple hosts can be specified separated by &. This can be useful where multiple MySQL servers are configured for replication and you are not concerned about the precise server you are connecting to. No attempt is made by the provider to synchronize writes to the database so care should be taken when using this option. In Unix environment with Mono, this can be a fully qualified path to MySQL socket filename. With this configuration, the Unix socket will be used instead of TCP/IP socket. Currently only a single socket name can be given so accessing MySQL in a replicated environment using Unix sockets is not currently supported.
Ignore Prepare	true	When true, instructs the provider to ignore any calls to MySQLCommand.Prepare() . This option is provided to prevent issues with corruption of the statements when use with server side prepared statements. If you want to use server-side prepare statements, set this option to false. This option was added in Connector/NET 5.0.3 and Connector/NET 1.0.9.
Port	3306	The port MySQL is using to listen for connections. Specify -1 for this value to use a named pipe connection (Windows only). This value is ignored if Unix socket is used.
Protocol	socket	Specifies the type of connection to make to the server. Values can be: socket or tcp for a socket connection pipe for a named pipe connection unix for a Unix socket connection memory to use MySQL shared memory
CharSet , Character Set		Specifies the character set that should be used to encode all queries sent to the server. Resultsets are still returned in the character set of the data returned.
Logging	false	When true, various pieces of information is output to any configured TraceListeners .
Allow Batch	true	When true, multiple SQL statements can be sent with one command exe-

		cution. -Note- Starting with MySQL 4.1.1, batch statements should be separated by the server-defined separator character. Commands sent to earlier versions of MySQL should be separated with ';'.
Encrypt	false	For Connector/NET 5.0.3 and later, when true , SSL encryption is used for all data sent between the client and server if the server has a certificate installed. Recognized values are true , false , yes , and no . In versions before 5.0.3, this option had no effect.
Initial Catalog, Database	mysql	The name of the database to use initially
Password, pwd		The password for the MySQL account being used.
Persist Security Info	false	When set to false or no (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values including the password. Recognized values are true , false , yes , and no .
User Id, Username, Uid, User name		The MySQL login account being used.
Shared Memory Name	MYSQL	The name of the shared memory object to use for communication if the connection protocol is set to memory.
Allow Zero Datetime	false	True to have <code>MySqlDataReader.GetValue()</code> return a <code>MySqlDateTime</code> for date or datetime columns that have illegal values. False will cause a <code>System.DateTime</code> object to be returned for legal values and an exception will be thrown for illegal values.
Convert Zero Datetime	false	True to have <code>MySqlDataReader.GetValue()</code> and <code>MySqlDataReader.GetDateTime()</code> return <code>DateTime.MinValue</code> for date or datetime columns that have illegal values.
Old Syntax, OldSyntax	false	Allows use of '@' symbol as a parameter marker. See <code>MySqlCommand</code> for more info. This is for compatibility only. All future code should be written to use the new '?' parameter marker.
Pipe Name, Pipe	mysql	When set to the name of a named pipe, the <code>MySqlConnection</code> will attempt to connect to MySQL on that named pipe. This settings only applies to the Windows platform.
Procedure Cache	25	Sets the size of the stored procedure cache. By default, Connector/NET will store the metadata (input/output datatypes) about the last 25 stored procedures used. To disable the stored procedure cache, set the value to zero (0). This option was added in Connector/NET 5.0.2 and Connector/NET 1.0.9.
Use Procedure Bodies	true	Setting this option to false indicates that the user connecting to the database does not have the <code>SELECT</code> privileges for the <code>mysql.proc</code> (stored procedures) table. When set to false , Connector/NET will not rely on this information being available when the procedure is called. Because Connector/NET will be unable to determine this information, you should explicitly set the types of all the parameters before the call and the parameters should be added to the command in the exact same order as they appear in the procedure definition. This option was added in Connector/NET 5.0.4 and Connector/NET 1.0.10.

The following table lists the valid names for connection pooling values within the `ConnectionString`. For more information about connection pooling, see Connection Pooling for the MySQL Data Provider.

Name	Default	Description
Connection Lifetime	0	When a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by <code>Connection Lifetime</code> . This is useful in clustered configurations to force load balancing between a running server and a server just brought online. A value of zero (0) causes pooled connections to have the maximum connection timeout.
Max Pool Size	100	The maximum number of connections allowed in the pool.

Min Pool Size	0	The minimum number of connections allowed in the pool.
Pooling	true	When true , the MySQLConnection object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool. Recognized values are true , false , yes , and no .
Reset Pooled Connections, ResetConnections, ResetPooledConnections	true	Specifies whether a ping and a reset should be sent to the server before a pooled connection is returned. Not resetting will yeild faster connection opens but also will not clear out session items such as temp tables.
Cache Server Configuration, CacheServerConfiguration, CacheServerConfig	false	Specifies whether server variables should be updated when a pooled connection is returned. Turning this one will yeild faster opens but will also not catch any server changes made by other connections.

When setting keyword or connection pooling values that require a Boolean value, you can use 'yes' instead of 'true', and 'no' instead of 'false'.

Note The MySQL Data Provider uses the native socket protocol to communicate with MySQL. Therefore, it does not support the use of an ODBC data source name (DSN) when connecting to MySQL because it does not add an ODBC layer.

CAUTION In this release, the application should use caution when constructing a connection string based on user input (for example when retrieving user ID and password information from a dialog box, and appending it to the connection string). The application should ensure that a user cannot embed extra connection string parameters in these values (for example, entering a password as "validpassword;database=somedb" in an attempt to attach to a different database).

Examples

The following example creates a **MySQLConnection** and sets some of its properties

Visual Basic example:

```
Public Sub CreateConnection()
    Dim myConnection As New MySQLConnection()
    myConnection.ConnectionString = "Persist Security Info=False;database=myDB;server=myHost;Connect Timeout=30;user id=myUser;password=myPass"
    myConnection.Open()
End Sub 'CreateConnection
```

C# example:

```
public void CreateConnection()
{
    MySQLConnection myConnection = new MySQLConnection();
    myConnection.ConnectionString = "Persist Security Info=False;database=myDB;server=myHost;Connect Timeout=30;user id=myUser;password=myPass"
    myConnection.Open();
}
```

Examples

The following example creates a **MySQLConnection** in Unix environment with Mono installed. MySQL socket filename used in this example is "/var/lib/mysql/mysql.sock". The actual filename depends on your MySQL configuration.

Visual Basic example:

```
Public Sub CreateConnection()
    Dim myConnection As New MySQLConnection()
    myConnection.ConnectionString = "database=myDB;server=/var/lib/mysql/mysql.sock;user id=myUser; pwd=myPass"
    myConnection.Open()
End Sub 'CreateConnection
```

C# example:

```
public void CreateConnection()
{
    MySQLConnection myConnection = new MySQLConnection();
    myConnection.ConnectionString = "database=myDB;server=/var/lib/mysql/mysql.sock;user id=myUser; pwd=myPass";
    myConnection.Open();
}
```

26.2.3.4. Using `MySQLDataAdapter`

Represents a set of data commands and a database connection that are used to fill a dataset and update a MySQL database. This class cannot be inherited.

The `MySQLDataAdapter`, serves as a bridge between a `System.Data.DataSet` and MySQL for retrieving and saving data. The `MySQLDataAdapter` provides this bridge by mapping `DbDataAdapter.Fill`, which changes the data in the `DataSet` to match the data in the data source, and `DbDataAdapter.Update`, which changes the data in the data source to match the data in the `DataSet`, using the appropriate SQL statements against the data source.

When the `MySQLDataAdapter` fills a `DataSet`, it will create the necessary tables and columns for the returned data if they do not already exist. However, primary key information will not be included in the implicitly created schema unless the `System.Data.MissingSchemaAction` property is set to `System.Data.MissingSchemaAction.AddWithKey`. You may also have the `MySQLDataAdapter` create the schema of the `DataSet`, including primary key information, before filling it with data using `System.Data.Common.DbDataAdapter.FillSchema`.

`MySQLDataAdapter` is used in conjunction with `MySQLConnection` and `MySQLCommand` to increase performance when connecting to a MySQL database.

The `MySQLDataAdapter` also includes the `MySQLDataAdapter.SelectCommand`, `MySQLDataAdapter.InsertCommand`, `MySQLDataAdapter.DeleteCommand`, `MySQLDataAdapter.UpdateCommand`, and `DataAdapter.TableMappings` properties to facilitate the loading and updating of data.

When an instance of `MySQLDataAdapter` is created, the read/write properties are set to initial values. For a list of these values, see the `MySQLDataAdapter` constructor.

Note

Please be aware that the `DataColumn` class in .NET 1.0 and 1.1 does not allow columns with type of `UInt16`, `UInt32`, or `UInt64` to be autoincrement columns. If you plan to use autoincrement columns with MySQL, you should consider using signed integer columns.

Examples

The following example creates a `MySQLCommand` and a `MySQLConnection`. The `MySQLConnection` is opened and set as the `MySQLCommand.Connection` for the `MySQLCommand`. The example then calls `MySQLCommand.ExecuteNonQuery`, and closes the connection. To accomplish this, the `ExecuteNonQuery` is passed a connection string and a query string that is a SQL INSERT statement.

Visual Basic example:

```
Public Function SelectRows(dataset As DataSet, connection As String, query As String) As DataSet
    Dim conn As New MySQLConnection(connection)
    Dim adapter As New MySQLDataAdapter()
    adapter.SelectCommand = new MySQLCommand(query, conn)
    adapter.Fill(dataset)
    Return dataset
End Function
```

C# example:

```
public DataSet SelectRows(DataSet dataset, string connection, string query)
{
    MySQLConnection conn = new MySQLConnection(connection);
    MySQLDataAdapter adapter = new MySQLDataAdapter();
    adapter.SelectCommand = new MySQLCommand(query, conn);
    adapter.Fill(dataset);
    return dataset;
}
```

26.2.3.4.1. Class `MySQLDataAdapter` Constructor

Overload methods for `MySQLDataAdapter`

Initializes a new instance of the `MySQLDataAdapter` class.

When an instance of [MySQLDataAdapter](#) is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

You can change the value of any of these properties through a separate call to the property.

Examples

The following example creates a [MySQLDataAdapter](#) and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlDataAdapter()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=localhost;" & _
        "database=test")
    Dim da As MySQLDataAdapter = New MySQLDataAdapter
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.SelectCommand = New MySqlCommand("SELECT id, name FROM mytable", conn)
    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
End Sub
```

C# example:

```
public static void CreateSqlDataAdapter()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost;database=test");
    MySQLDataAdapter da = new MySQLDataAdapter();
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.SelectCommand = new MySqlCommand("SELECT id, name FROM mytable", conn);
    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
}
```

26.2.3.4.2. Class MySQLDataAdapter Constructor Form 1

Initializes a new instance of the [MySQLDataAdapter](#) class with the specified [MySqlCommand](#) as the [SelectCommand](#) property.

Parameters: [MySqlCommand](#) that is a SQL [SELECT](#) statement or stored procedure and is set as the [SelectCommand](#) property of the [MySQLDataAdapter](#).

When an instance of [MySQLDataAdapter](#) is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

You can change the value of any of these properties through a separate call to the property.

When [SelectCommand](#) (or any of the other command properties) is assigned to a previously created [MySQLCommand](#), the [MySQLCommand](#) is not cloned. The [SelectCommand](#) maintains a reference to the previously created [MySQLCommand](#) object.

Examples

The following example creates a [MySQLDataAdapter](#) and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlDataAdapter()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=localhost;" & _
        "database=test")
    Dim cmd As new MySqlCommand("SELECT id, name FROM mytable", conn)
    Dim da As MySQLDataAdapter = New MySQLDataAdapter(cmd)
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
End Sub
```

C# example:

```
public static void CreateSqlDataAdapter()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost;database=test");
    MySqlCommand cmd = new MySqlCommand("SELECT id, name FROM mytable", conn);
    MySQLDataAdapter da = new MySQLDataAdapter(cmd);
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
}
```

26.2.3.4.3. Class MySQLDataAdapter Constructor Form 2

Initializes a new instance of the [MySQLDataAdapter](#) class with a [SelectCommand](#) and a [MySqlConnection](#) object.

Parameters: A [String](#) that is a SQL [SELECT](#) statement or stored procedure to be used by the [SelectCommand](#) property of the [MySQLDataAdapter](#).

Parameters: A [MySqlConnection](#) that represents the connection.

This implementation of the [MySQLDataAdapter](#) opens and closes a [MySqlConnection](#) if it is not already open. This can be useful in an application that must call the [DbDataAdapter.Fill](#) method for two or more [MySQLDataAdapter](#) objects. If the [MySqlConnection](#) is already open, you must explicitly call [MySqlConnection.Close](#) or [MySqlConnection.Dispose](#) to close it.

When an instance of [MySQLDataAdapter](#) is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
MissingMappingAction	MissingMappingAction.Passthrough

MissingSchemaAction	MissingSchemaAction.Add

You can change the value of any of these properties through a separate call to the property.

Examples

The following example creates a [MySQLDataAdapter](#) and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlDataAdapter()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=localhost;" & _
        "database=test")
    Dim da As MySqlDataAdapter = New MySqlDataAdapter("SELECT id, name FROM mytable", conn)
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
End Sub
```

C# example:

```
public static void CreateSqlDataAdapter()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost;database=test");
    MySqlDataAdapter da = new MySqlDataAdapter("SELECT id, name FROM mytable", conn);
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
}
```

26.2.3.4.4. Class MySqlDataAdapter Constructor Form 3

Initializes a new instance of the [MySqlDataAdapter](#) class with a [SelectCommand](#) and a connection string.

Parameters: A [string](#) that is a SQL [SELECT](#) statement or stored procedure to be used by the [SelectCommand](#) property of the [MySqlDataAdapter](#).

Parameters: The connection string

When an instance of [MySqlDataAdapter](#) is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

You can change the value of any of these properties through a separate call to the property.

Examples

The following example creates a [MySQLDataAdapter](#) and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlDataAdapter()
    Dim da As MySQLDataAdapter = New MySQLDataAdapter("SELECT id, name FROM mytable", "Data Source=localhost;database=test")
    Dim conn As MySqlConnection = da.SelectCommand.Connection
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySQLDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySQLDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySQLDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySQLDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySQLDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
    da.DeleteCommand.Parameters.Add("?id", MySQLDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
End Sub
```

C# example:

```
public static void CreateSqlDataAdapter()
{
    MySQLDataAdapter da = new MySQLDataAdapter("SELECT id, name FROM mytable", "Data Source=localhost;database=test");
    MySqlConnection conn = da.SelectCommand.Connection;
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySQLDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySQLDbType.VarChar, 40, "name");

    da.UpdateCommand.Parameters.Add("?id", MySQLDbType.VarChar, 5, "id");
    da.UpdateCommand.Parameters.Add("?name", MySQLDbType.VarChar, 40, "name");
    da.UpdateCommand.Parameters.Add("?oldId", MySQLDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
    da.DeleteCommand.Parameters.Add("?id", MySQLDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
}
```

26.2.3.4.5. DeleteCommand

Gets or sets a SQL statement or stored procedure used to delete records from the data set.

Value: A [MySqlCommand](#) used during [System.Data.Common.DataAdapter.Update](#) to delete records in the database that correspond to deleted rows in the [DataSet](#).

During [System.Data.Common.DataAdapter.Update](#), if this property is not set and primary key information is present in the [DataSet](#), the [DeleteCommand](#) can be generated automatically if you set the [SelectCommand](#) property and use the [MySQLCommandBuilder](#). Then, any additional commands that you do not set are generated by the [MySQLCommandBuilder](#). This generation logic requires key column information to be present in the [DataSet](#).

When [DeleteCommand](#) is assigned to a previously created [MySqlCommand](#), the [MySqlCommand](#) is not cloned. The [DeleteCommand](#) maintains a reference to the previously created [MySqlCommand](#) object.

Examples

The following example creates a [MySQLDataAdapter](#) and sets the [SelectCommand](#) and [DeleteCommand](#) properties. It assumes you have already created a [MySQLConnection](#) object.

Visual Basic example:

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySQLDataAdapter
    Dim da As MySQLDataAdapter = New MySQLDataAdapter()
    Dim cmd As MySqlCommand
    Dim parm As MySQLParameter
    ' Create the SelectCommand.
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
    cmd.Parameters.Add("?id", MySQLDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySQLDbType.VarChar, 15)
```



```

da.SelectCommand = cmd
' Create the DeleteCommand.
cmd = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
parm = cmd.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
parm.SourceVersion = DataRowVersion.Original
da.DeleteCommand = cmd
Return da
End Function

```

C# example:

```

public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
    MySqlDataAdapter da = new MySqlDataAdapter();
    MySqlCommand cmd;
    MySqlParameter parm;
    // Create the SelectCommand.
    cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
    da.SelectCommand = cmd;
    // Create the DeleteCommand.
    cmd = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    parm = cmd.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    parm.SourceVersion = DataRowVersion.Original;
    da.DeleteCommand = cmd;
    return da;
}

```

26.2.3.4.6. InsertCommand

Gets or sets a SQL statement or stored procedure used to insert records into the data set.

Value: A [MySqlCommand](#) used during [System.Data.Common.DataAdapter.Update](#) to insert records into the database that correspond to new rows in the [DataSet](#).

During [System.Data.Common.DataAdapter.Update](#), if this property is not set and primary key information is present in the [DataSet](#), the [InsertCommand](#) can be generated automatically if you set the [SelectCommand](#) property and use the [MySqlCommandBuilder](#). Then, any additional commands that you do not set are generated by the [MySqlCommandBuilder](#). This generation logic requires key column information to be present in the [DataSet](#).

When [InsertCommand](#) is assigned to a previously created [MySqlCommand](#), the [MySqlCommand](#) is not cloned. The [InsertCommand](#) maintains a reference to the previously created [MySqlCommand](#) object.

Note

If execution of this command returns rows, these rows may be added to the [DataSet](#) depending on how you set the [MySqlCommand.UpdatedRowSource](#) property of the [MySqlCommand](#) object.

Examples

The following example creates a [MySqlDataAdapter](#) and sets the [SelectCommand](#) and [InsertCommand](#) properties. It assumes you have already created a [MySqlConnection](#) object.

Visual Basic example:

```

Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySqlDataAdapter
    Dim da As MySqlDataAdapter = New MySqlDataAdapter()
    Dim cmd As MySqlCommand
    Dim parm As MySqlParameter
    ' Create the SelectCommand.
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
    da.SelectCommand = cmd
    ' Create the InsertCommand.
    cmd = New MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id, ?name)", conn)
    cmd.Parameters.Add( "?id", MySqlDbType.VarChar, 15, "id" )
    cmd.Parameters.Add( "?name", MySqlDbType.VarChar, 15, "name" )
    da.InsertCommand = cmd

    Return da
End Function

```

C# example:

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
    MySqlDataAdapter da = new MySqlDataAdapter();
    MySqlCommand cmd;
    MySqlParameter parm;
    // Create the SelectCommand.
    cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
    da.SelectCommand = cmd;
    // Create the InsertCommand.
    cmd = new MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id,?name)", conn);
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id" );
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name" );

    da.InsertCommand = cmd;
    return da;
}
```

26.2.3.4.7. UpdateCommand

Gets or sets a SQL statement or stored procedure used to updated records in the data source.

Value: A [MySqlCommand](#) used during [System.Data.Common.DataAdapter.Update](#) to update records in the database with data from the [DataSet](#).

During [System.Data.Common.DataAdapter.Update](#), if this property is not set and primary key information is present in the [DataSet](#), the [UpdateCommand](#) can be generated automatically if you set the [SelectCommand](#) property and use the [MySqlCommandBuilder](#). Then, any additional commands that you do not set are generated by the [MySqlCommandBuilder](#). This generation logic requires key column information to be present in the [DataSet](#).

When [UpdateCommand](#) is assigned to a previously created [MySqlCommand](#), the [MySqlCommand](#) is not cloned. The [UpdateCommand](#) maintains a reference to the previously created [MySqlCommand](#) object.

Note

If execution of this command returns rows, these rows may be merged with the [DataSet](#) depending on how you set the [MySqlCommand.UpdatedRowSource](#) property of the [MySqlCommand](#) object.

Examples

The following example creates a [MySqlDataAdapter](#) and sets the [SelectCommand](#) and [UpdateCommand](#) properties. It assumes you have already created a [MySqlConnection](#) object.

Visual Basic example:

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySqlDataAdapter
    Dim da As MySqlDataAdapter = New MySqlDataAdapter()
    Dim cmd As MySqlCommand
    Dim parm As MySqlParameter
    ' Create the SelectCommand.
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
    da.SelectCommand = cmd
    ' Create the UpdateCommand.
    cmd = New MySqlCommand("UPDATE mytable SET id=?id, name=?name WHERE id=?oldId", conn)
    cmd.Parameters.Add( "?id", MySqlDbType.VarChar, 15, "id" )
    cmd.Parameters.Add( "?name", MySqlDbType.VarChar, 15, "name" )

    parm = cmd.Parameters.Add("?oldId", MySqlDbType.VarChar, 15, "id")
    parm.SourceVersion = DataRowVersion.Original

    da.UpdateCommand = cmd

    Return da
End Function
```

C# example:

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
    MySqlDataAdapter da = new MySqlDataAdapter();
```

```

MySQLCommand cmd;
MySQLParameter parm;
// Create the SelectCommand.
cmd = new MySQLCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
cmd.Parameters.Add("?id", MySQLDbType.VarChar, 15);
cmd.Parameters.Add("?name", MySQLDbType.VarChar, 15);
da.SelectCommand = cmd;
// Create the UpdateCommand.
cmd = new MySQLCommand("UPDATE mytable SET id=?id, name=?name WHERE id=?oldId", conn);
cmd.Parameters.Add("?id", MySQLDbType.VarChar, 15, "id" );
cmd.Parameters.Add("?name", MySQLDbType.VarChar, 15, "name" );

parm = cmd.Parameters.Add( "?oldId", MySQLDbType.VarChar, 15, "id" );
parm.SourceVersion = DataRowVersion.Original;

da.UpdateCommand = cmd;
return da;
}

```

26.2.3.4.8. SelectCommand

Gets or sets a SQL statement or stored procedure used to select records in the data source.

Value: A [MySQLCommand](#) used during [System.Data.Common.DbDataAdapter.Fill](#) to select records from the database for placement in the [DataSet](#).

When [SelectCommand](#) is assigned to a previously created [MySQLCommand](#), the [MySQLCommand](#) is not cloned. The [SelectCommand](#) maintains a reference to the previously created [MySQLCommand](#) object.

If the [SelectCommand](#) does not return any rows, no tables are added to the [DataSet](#), and no exception is raised.

Examples

The following example creates a [MySQLDataAdapter](#) and sets the [SelectCommand](#) and [InsertCommand](#) properties. It assumes you have already created a [MySQLConnection](#) object.

Visual Basic example:

```

Public Shared Function CreateCustomerAdapter(conn As MySQLConnection) As MySQLDataAdapter

    Dim da As MySQLDataAdapter = New MySQLDataAdapter()
    Dim cmd As MySQLCommand
    Dim parm As MySQLParameter
    ' Create the SelectCommand.
    cmd = New MySQLCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
    cmd.Parameters.Add("?id", MySQLDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySQLDbType.VarChar, 15)
    da.SelectCommand = cmd
    ' Create the InsertCommand.
    cmd = New MySQLCommand("INSERT INTO mytable (id,name) VALUES (?id, ?name)", conn)
    cmd.Parameters.Add( "?id", MySQLDbType.VarChar, 15, "id" )
    cmd.Parameters.Add( "?name", MySQLDbType.VarChar, 15, "name" )
    da.InsertCommand = cmd

    Return da
End Function

```

C# example:

```

public static MySQLDataAdapter CreateCustomerAdapter(MySQLConnection conn)
{
    MySQLDataAdapter da = new MySQLDataAdapter();
    MySQLCommand cmd;
    MySQLParameter parm;
    // Create the SelectCommand.
    cmd = new MySQLCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
    cmd.Parameters.Add("?id", MySQLDbType.VarChar, 15);
    cmd.Parameters.Add("?name", MySQLDbType.VarChar, 15);
    da.SelectCommand = cmd;
    // Create the InsertCommand.
    cmd = new MySQLCommand("INSERT INTO mytable (id,name) VALUES (?id,?name)", conn);
    cmd.Parameters.Add("?id", MySQLDbType.VarChar, 15, "id" );
    cmd.Parameters.Add("?name", MySQLDbType.VarChar, 15, "name" );

    da.InsertCommand = cmd;
    return da;
}

```

26.2.3.5. Using `MySqlDataReader`

To create a `MySqlDataReader`, you must call the `MySqlCommand.ExecuteReader` method of the `MySqlCommand` object, rather than directly using a constructor.

While the `MySqlDataReader` is in use, the associated `MySqlConnection` is busy serving the `MySqlDataReader`, and no other operations can be performed on the `MySqlConnection` other than closing it. This is the case until the `MySqlDataReader.Close` method of the `MySqlDataReader` is called.

`MySqlDataReader.IsClosed` and `MySqlDataReader.RecordsAffected` are the only properties that you can call after the `MySqlDataReader` is closed. Though the `RecordsAffected` property may be accessed at any time while the `MySqlDataReader` exists, always call `Close` before returning the value of `RecordsAffected` to ensure an accurate return value.

For optimal performance, `MySqlDataReader` avoids creating unnecessary objects or making unnecessary copies of data. As a result, multiple calls to methods such as `MySqlDataReader.GetValue` return a reference to the same object. Use caution if you are modifying the underlying value of the objects returned by methods such as `GetValue`.

Examples

The following example creates a `MySqlConnection`, a `MySqlCommand`, and a `MySqlDataReader`. The example reads through the data, writing it out to the console. Finally, the example closes the `MySqlDataReader`, then the `MySqlConnection`.

Visual Basic example:

```
Public Sub ReadMyData(myConnString As String)
    Dim mySelectQuery As String = "SELECT OrderID, CustomerID FROM Orders"
    Dim myConnection As New MySqlConnection(myConnString)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myConnection.Open()
    Dim myReader As MySqlDataReader
    myReader = myCommand.ExecuteReader()
    ' Always call Read before accessing data.
    While myReader.Read()
        Console.WriteLine((myReader.GetInt32(0) & ", " & myReader.GetString(1)))
    End While
    ' always call Close when done reading.
    myReader.Close()
    ' Close the connection when done with it.
    myConnection.Close()
End Sub 'ReadMyData
```

C# example:

```
public void ReadMyData(string myConnString) {
    string mySelectQuery = "SELECT OrderID, CustomerID FROM Orders";
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnection);
    myConnection.Open();
    MySqlDataReader myReader;
    myReader = myCommand.ExecuteReader();
    // Always call Read before accessing data.
    while (myReader.Read()) {
        Console.WriteLine(myReader.GetInt32(0) + ", " + myReader.GetString(1));
    }
    // always call Close when done reading.
    myReader.Close();
    // Close the connection when done with it.
    myConnection.Close();
}
```

26.2.3.5.1. `GetBytes`

`GetBytes` returns the number of available bytes in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if `GetBytes` has already been used to obtain bytes from the field. This may be the case, for example, if the `MySqlDataReader` is reading a large data structure into a buffer. For more information, see the `SequentialAccess` setting for `MySqlCommand.CommandBehavior`.

If you pass a buffer that is a null reference (`Nothing` in Visual Basic), `GetBytes` returns the length of the field in bytes.

No conversions are performed; therefore the data retrieved must already be a byte array.

26.2.3.5.2. GetTimeSpan

Gets the value of the specified column as a [TimeSpan](#) object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.3. GetDateTime

Gets the value of the specified column as a [System.DateTime](#) object.

Note

MySQL allows date columns to contain the value '0000-00-00' and datetime columns to contain the value '0000-00-00 00:00:00'. The `DateTime` structure cannot contain or represent these values. To read a datetime value from a column that might contain zero values, use [GetMySQLDateTime](#). The behavior of reading a zero datetime column using this method is defined by the [ZeroDateTimeBehavior](#) connection string option. For more information on this option, please refer to [MySQLConnection.ConnectionString](#).

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.4. GetMySQLDateTime

Gets the value of the specified column as a [MySQL.Data.Types.MySQLDateTime](#) object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.5. GetString

Gets the value of the specified column as a [String](#) object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.6. GetDecimal

Gets the value of the specified column as a [Decimal](#) object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.7. GetDouble

Gets the value of the specified column as a double-precision floating point number.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.8. GetFloat

Gets the value of the specified column as a single-precision floating point number.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.9. GetGuid

Gets the value of the specified column as a globally-unique identifier (GUID).

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.10. GetInt16

Gets the value of the specified column as a 16-bit signed integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.11. GetInt32

Gets the value of the specified column as a 32-bit signed integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.12. GetInt64

Gets the value of the specified column as a 64-bit signed integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.13. GetUInt16

Gets the value of the specified column as a 16-bit unsigned integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.14. GetUInt32

Gets the value of the specified column as a 32-bit unsigned integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.5.15. GetUInt64

Gets the value of the specified column as a 64-bit unsigned integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

26.2.3.6. Using `MySqlException`

This class is created whenever the MySQL Data Provider encounters an error generated from the server.

Any open connections are not automatically closed when an exception is thrown. If the client application determines that the exception is fatal, it should close any open `MySqlDataReader` objects or `MySqlConnection` objects.

Examples

The following example generates a [MySQLException](#) due to a missing server, and then displays the exception.

Visual Basic example:

```
Public Sub ShowException()  
    Dim mySelectQuery As String = "SELECT column1 FROM table1"  
    Dim myConnection As New MySqlConnection ("Data Source=localhost;Database=Sample;")  
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)  
    Try  
        myCommand.Connection.Open()  
    Catch e As MySQLException  
        MessageBox.Show( e.Message )  
    End Try  
End Sub
```

C# example:

```
public void ShowException()  
{  
    string mySelectQuery = "SELECT column1 FROM table1";  
    MySqlConnection myConnection =  
        new MySqlConnection("Data Source=localhost;Database=Sample;");  
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery,myConnection);  
    try  
    {  
        myCommand.Connection.Open();  
    }  
    catch (MySQLException e)  
    {  
        MessageBox.Show( e.Message );  
    }  
}
```

26.2.3.7. Using [MySQLParameter](#)

Parameter names are not case sensitive.

Examples

The following example creates multiple instances of [MySQLParameter](#) through the [MySQLParameterCollection](#) collection within the [MySQLDataAdapter](#). These parameters are used to select data from the data source and place the data in the [DataSet](#). This example assumes that a [DataSet](#) and a [MySQLDataAdapter](#) have already been created with the appropriate schema, commands, and connection.

Visual Basic example:

```
Public Sub AddSqlParameters()  
    ' ...  
    ' create myDataSet and myDataAdapter  
    ' ...  
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySQLDbType.VarChar, 80).Value = "toasters"  
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySQLDbType.Long).Value = 239  
    myDataAdapter.Fill(myDataSet)  
End Sub 'AddSqlParameters
```

C# example:

```
public void AddSqlParameters()  
{  
    // ...  
    // create myDataSet and myDataAdapter  
    // ...  
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySQLDbType.VarChar, 80).Value = "toasters";  
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySQLDbType.Long).Value = 239;  
    myDataAdapter.Fill(myDataSet);  
}
```

26.2.3.8. Using [MySQLParameterCollection](#)

The number of the parameters in the collection must be equal to the number of parameter placeholders within the command text, or an exception will be generated.

Examples

The following example creates multiple instances of [MySQLParameter](#) through the [MySQLParameterCollection](#) collection within the [MySQLDataAdapter](#). These parameters are used to select data within the data source and place the data in the [DataSet](#). This code assumes that a [DataSet](#) and a [MySQLDataAdapter](#) have already been created with the appropriate schema, commands, and connection.

Visual Basic example:

```
Public Sub AddParameters()
    ' ...
    ' create myDataSet and myDataAdapter
    ' ...
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar, 80).Value = "toasters"
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Long).Value = 239
    myDataAdapter.Fill(myDataSet)
End Sub 'AddSqlParameters
```

C# example:

```
public void AddSqlParameters()
{
    // ...
    // create myDataSet and myDataAdapter
    // ...
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar, 80).Value = "toasters";
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Long).Value = 239;
    myDataAdapter.Fill(myDataSet);
}
```

26.2.3.9. Using [MySQLTransaction](#)

Represents a SQL transaction to be made in a MySQL database. This class cannot be inherited.

The application creates a [MySQLTransaction](#) object by calling [MySQLConnection.BeginTransaction](#) on the [MySQLConnection](#) object. All subsequent operations associated with the transaction (for example, committing or aborting the transaction), are performed on the [MySQLTransaction](#) object.

Examples

The following example creates a [MySQLConnection](#) and a [MySQLTransaction](#). It also demonstrates how to use the [MySQLConnection.BeginTransaction](#), [MySQLTransaction.Commit](#), and [MySQLTransaction.Rollback](#) methods.

Visual Basic example:

```
Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySQLTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (100, 'Description')"
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (101, 'Description')"
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Both records are written to database.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySQLException
```



```

        If Not myTrans.Connection Is Nothing Then
            Console.WriteLine("An exception of type " & ex.GetType().ToString() & _
                " was encountered while attempting to roll back the transaction.")
        End If
    End Try

    Console.WriteLine("An exception of type " & e.GetType().ToString() & _
        " was encountered while inserting the data.")
    Console.WriteLine("Neither record was written to database.")
Finally
    myConnection.Close()
End Try
End Sub 'RunTransaction

```

C# example:

```

public void RunTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (100, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (101, 'Description')";
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch (Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (MySqlException ex)
        {
            if (myTrans.Connection != null)
            {
                Console.WriteLine("An exception of type " + ex.GetType() +
                    " was encountered while attempting to roll back the transaction.");
            }
        }

        Console.WriteLine("An exception of type " + e.GetType() +
            " was encountered while inserting the data.");
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
        myConnection.Close();
    }
}

```

26.2.3.9.1. Rollback

Rolls back a transaction from a pending state.

The Rollback method is equivalent to the MySQL statement ROLLBACK. The transaction can only be rolled back from a pending state (after BeginTransaction has been called, but before Commit is called).

Examples

The following example creates [MySqlConnection](#) and a [MySqlTransaction](#). It also demonstrates how to use the [MySqlConnection.BeginTransaction](#), [Commit](#), and [Rollback](#) methods.

Visual Basic example:

```

Public Sub RunSqlTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

```

```

Dim myCommand As MySqlCommand = myConnection.CreateCommand()
Dim myTrans As MySqlTransaction

' Start a local transaction
myTrans = myConnection.BeginTransaction()

' Must assign both transaction object and connection
' to Command object for a pending local transaction
myCommand.Connection = myConnection
myCommand.Transaction = myTrans

Try
    myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')"
    myCommand.ExecuteNonQuery()
    myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')"
    myCommand.ExecuteNonQuery()
    myTrans.Commit()
    Console.WriteLine("Success.")
Catch e As Exception
    Try
        myTrans.Rollback()
    Catch ex As MySqlException
        If Not myTrans.Connection Is Nothing Then
            Console.WriteLine("An exception of type " & ex.GetType().ToString() & _
                " was encountered while attempting to roll back the transaction.")
        End If
    End Try

    Console.WriteLine("An exception of type " & e.GetType().ToString() & _
        " was encountered while inserting the data.")
    Console.WriteLine("Neither record was written to database.")
Finally
    myConnection.Close()
End Try
End Sub

```

C# example:

```

public void RunSqlTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')";
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch (Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (MySqlException ex)
        {
            if (myTrans.Connection != null)
            {
                Console.WriteLine("An exception of type " + ex.GetType() +
                    " was encountered while attempting to roll back the transaction.");
            }
        }

        Console.WriteLine("An exception of type " + e.GetType() +
            " was encountered while inserting the data.");
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
        myConnection.Close();
    }
}

```

26.2.3.9.2. Commit

Commits the database transaction.

The [Commit](#) method is equivalent to the MySQL SQL statement COMMIT.

Examples

The following example creates [MySqlConnection](#) and a [MySqlTransaction](#). It also demonstrates how to use the [MySqlConnection.BeginTransaction](#), [Commit](#), and [Rollback](#) methods.

Visual Basic example:

```
Public Sub RunSqlTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()

    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')"
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')"
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Success.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySqlException
            If Not myTrans.Connection Is Nothing Then
                Console.WriteLine("An exception of type " & ex.GetType().ToString() & _
                    " was encountered while attempting to roll back the transaction.")
            End If
        End Try

        Console.WriteLine("An exception of type " & e.GetType().ToString() & _
            " was encountered while inserting the data.")
        Console.WriteLine("Neither record was written to database.")
    Finally
        myConnection.Close()
    End Try
End Sub
```

C# example:

```
public void RunSqlTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')";
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (MySqlException ex)
        {
        }
    }
}
```

```

    {
        if (myTrans.Connection != null)
        {
            Console.WriteLine("An exception of type " + ex.GetType() +
                " was encountered while attempting to roll back the transaction.");
        }
        Console.WriteLine("An exception of type " + e.GetType() +
            " was encountered while inserting the data.");
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
        myConnection.Close();
    }
}

```

26.2.4. Connector/NET Reference

This section of the manual contains a complete reference to the Connector/NET ADO.NET component, automatically generated from the embedded documentation.

26.2.4.1. `MySql.Data.MySqlClient`

[Namespace hierarchy](#)

Classes

Class	Description
 MySqlCommand 	
 MySqlCommandBuilder 	
 MySqlConnection 	
 MySqlDataAdapter 	
 MySqlDataReader 	Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.
 MySqlError 	Collection of error codes that can be returned by the server
 MySqlException 	The exception that is thrown when MySQL returns an error. This class cannot be inherited.
 MySqlHelper 	Helper class that makes it easier to work with the provider.
 MySqlInfoMessageEventArgs 	Provides data for the InfoMessage event. This class cannot be inherited.
 MySqlParameter 	Represents a parameter to a MySqlCommand , and optionally, its mapping to DataSet columns. This class cannot be inherited.
 MySqlParameterCollection 	Represents a collection of parameters relevant to a MySqlCommand as well as their respective mappings to columns in a DataSet. This class cannot be inherited.
 MySqlRowUpdatedEventArgs 	Provides data for the RowUpdated event. This class cannot be inherited.
 MySqlRowUpdatingEventArgs 	Provides data for the RowUpdating event. This class cannot be inherited.
 MySqlTransaction 	

Delegates

Delegate	Description
 MySqlInfoMessageEventHandler 	Represents the method that will handle the InfoMessage event of a MySqlConnection .
 MySqlRowUpdatedEventHandler 	Represents the method that will handle the RowUpdated event of a MySqlDataAdapter .

MySqlRowUpdatingEventHandler	Represents the method that will handle the RowUpdatingevent of a MySqlDataAdapter .

Enumerations

Enumeration	Description
MySqlDbType	Specifies MySQL specific data type of a field, property, for use in a MySqlParameter .
MySqlErrorCode	

26.2.4.1.1. [MySQL.Data.MySqlCommandHierarchy](#)**See Also**

[MySQL.Data.MySqlCommand Namespace](#)

26.2.4.1.2. [MySQLCommand Class](#)

For a list of all members of this type, see [MySQLCommand Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommand_
    Inherits Component_
    Implements IDbCommand, ICloneable
```

Syntax: C#

```
public sealed class MySqlCommand : Component, IDbCommand, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlCommand](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLCommand Members](#) , [MySQL.Data.MySqlCommand Namespace](#)

26.2.4.1.2.1. [MySQLCommand Members](#)

[MySQLCommand overview](#)

Public Instance Constructors

MySQLCommand	Overloaded. Initializes a new instance of the MySqlCommand class.

Public Instance Properties

CommandText	
CommandTimeout	
CommandType	

Connection	
Container(inherited from Component)	Gets the IContainerthat contains the Component.
IsPrepared	
Parameters	
Site(inherited from Component)	Gets or sets the ISiteof the Component.
Transaction	
UpdatedRowSource	

Public Instance Methods

Cancel	Attempts to cancel the execution of a MySqlCommand. This operation is not supported.
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
CreateParameter	Creates a new instance of a MySqlParameter object.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
ExecuteNonQuery	
ExecuteReader	Overloaded.
ExecuteScalar	
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Typeof the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Prepare	
ToString(inherited from Component)	Returns a Stringcontaining the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposedevent on the component.

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1. MySqlCommand Constructor

Initializes a new instance of the [MySqlCommand](#) class.

Overload List

Initializes a new instance of the [MySqlCommand](#) class.

- [public MySqlCommand\(\);](#)

- [public MySqlCommand\(string\);](#)
- [public MySqlCommand\(string, MySqlConnection\);](#)
- [public MySqlCommand\(string, MySqlConnection, MySqlTransaction\);](#)

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.1. MySqlCommand Constructor ()

Initializes a new instance of the [MySqlCommand](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlCommand();
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlCommand Constructor Overload List](#)

26.2.4.1.2.1.1.2. MySqlCommand Constructor (String)

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String _  
)
```

Syntax: C#

```
public MySqlCommand(  
    stringcmdText  
);
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlCommand Constructor Overload List](#)

26.2.4.1.2.1.1.3. MySqlCommand Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String, _  
    ByVal connection As MySqlConnection _  
)
```

Syntax: C#

```
public MySqlCommand(  
    stringcmdText,  
    MySqlConnectionconnection  
);
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlCommand Constructor Overload List](#)

26.2.4.1.2.1.1.3.1. MySqlConnection Class

For a list of all members of this type, see [MySQLConnection Members](#).

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnection_
    Inherits Component_
    Implements IDbConnection, ICloneable
```

Syntax: C#

```
public sealed class MySqlConnection : Component, IDbConnection, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLConnection Members](#), [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1. MySqlConnection Members

[MySQLConnection overview](#)

Public Instance Constructors

MySQLConnection	Overloaded. Initializes a new instance of the MySqlConnection class.

Public Instance Properties

ConnectionString	
ConnectionTimeout	
Container(inherited from Component)	Gets the IContainerthat contains the Component.
Database	
DataSource	Gets the name of the MySQL server to which to connect.
ServerThread	Returns the id of the server thread this connection is executing on
ServerVersion	
Site(inherited from Component)	Gets or sets the ISiteof the Component.
State	
UseCompression	Indicates if this connection should use compression when communicating with the server.

Public Instance Methods

BeginTransaction	Overloaded.
ChangeDatabase	
Close	
CreateCommand	
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information re-

	quired to generate a proxy used to communicate with a remote object.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Open	
Ping	Ping
ToString(inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
InfoMessage	
StateChange	

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.1. MySQLConnection Constructor

Initializes a new instance of the [MySQLConnection](#) class.

Overload List

Initializes a new instance of the [MySQLConnection](#) class.

- [public MySQLConnection\(\);](#)
- [public MySQLConnection\(string\);](#)

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.1.1. MySQLConnection Constructor

Initializes a new instance of the [MySQLConnection](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySQLConnection();
```

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLConnection Constructor Overload List](#)

26.2.4.1.2.1.1.3.1.1.2. **MySQLConnection** Constructor**Syntax: Visual Basic**

```
Overloads Public Sub New( _  
    ByVal connectionString As String _  
)
```

Syntax: C#

```
public MySqlConnection(  
    string connectionString  
);
```

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLConnection Constructor Overload List](#)

26.2.4.1.2.1.1.3.1.1.2. **ConnectionString** Property**Syntax: Visual Basic**

```
NotOverridable Public Property ConnectionString As String _  
    Implements IDbConnection.ConnectionString
```

Syntax: C#

```
public string ConnectionString {get; set;}
```

Implements

IDbConnection.ConnectionString

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.3. **ConnectionTimeout** Property**Syntax: Visual Basic**

```
NotOverridable Public ReadOnly Property ConnectionTimeout As Integer _  
    Implements IDbConnection.ConnectionTimeout
```

Syntax: C#

```
public int ConnectionTimeout {get;}
```

Implements

IDbConnection.ConnectionTimeout

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.4. **Database** Property**Syntax: Visual Basic**

```
NotOverridable Public ReadOnly Property Database As String _  
    Implements IDbConnection.Database
```

Syntax: C#

```
public string Database {get;}
```

Implements

IDbConnection.Database

See Also

[MySqlConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.5. DataSource Property

Gets the name of the MySQL server to which to connect.

Syntax: Visual Basic

```
Public ReadOnly Property DataSource As String
```

Syntax: C#

```
public string DataSource {get;}
```

See Also

[MySqlConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.6. ServerThread Property

Returns the id of the server thread this connection is executing on

Syntax: Visual Basic

```
Public ReadOnly Property ServerThread As Integer
```

Syntax: C#

```
public int ServerThread {get;}
```

See Also

[MySqlConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.7. ServerVersion Property

Syntax: Visual Basic

```
Public ReadOnly Property ServerVersion As String
```

Syntax: C#

```
public string ServerVersion {get;}
```

See Also

[MySqlConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.8. State Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property State As ConnectionState _  
    Implements IDbConnection.State
```

Syntax: C#

```
public System.Data.ConnectionState State {get;}
```

Implements

IDbConnection.State

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.9. UseCompression Property

Indicates if this connection should use compression when communicating with the server.

Syntax: Visual Basic

```
Public ReadOnly Property UseCompression As Boolean
```

Syntax: C#

```
public bool UseCompression {get;}
```

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.10. BeginTransaction Method

Overload List

- [public MySQLTransaction BeginTransaction\(\);](#)
- [public MySQLTransaction BeginTransaction\(IsolationLevel\);](#)

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.10.1. [MySQLConnection.BeginTransaction](#) Method

Syntax: Visual Basic

```
Overloads Public Function BeginTransaction() As MySQLTransaction
```

Syntax: C#

```
public MySQLTransaction BeginTransaction();
```

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLConnection.BeginTransaction Overload List](#)

26.2.4.1.2.1.1.3.1.1.10.1.1. [MySQLTransaction](#) Class

For a list of all members of this type, see [MySQLTransaction Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySQLTransaction_
    Implements IDbTransaction, IDisposable
```

Syntax: C#

```
public sealed class MySQLTransaction : IDbTransaction, IDisposable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySQLClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLTransaction Members](#) , [MySQL.Data.MySQLClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.10.1.1.1.1. MySQLTransaction Members

[MySQLTransaction overview](#)

Public Instance Properties

Connection	Gets the MySQLConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.
IsolationLevel	Specifies the IsolationLevel for this transaction.

Public Instance Methods

Commit	
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Type of the current instance.
Rollback	
ToString(inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLTransaction Class](#) , [MySQL.Data.MySQLClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.10.1.1.1.1.1. Connection Property

Gets the [MySQLConnection](#) object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Syntax: Visual Basic

```
Public ReadOnly Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get;}
```

Property Value

The [MySqlConnection](#) object associated with this transaction.

Remarks

A single application may have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction created by [BeginTransaction](#).

See Also

[MySqlTransaction Class](#), [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.10.1.1.1.2. IsolationLevel Property

Specifies the IsolationLevel for this transaction.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsolationLevel As IsolationLevel _  
    Implements IDbTransaction.IsolationLevel
```

Syntax: C#

```
public System.Data.IsolationLevel IsolationLevel {get;}
```

Property Value

The IsolationLevel for this transaction. The default is ReadCommitted.

Implements

IDbTransaction.IsolationLevel

Remarks

Parallel transactions are not supported. Therefore, the IsolationLevel applies to the entire transaction.

See Also

[MySqlTransaction Class](#), [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.10.1.1.1.3. [MySqlTransaction.Commit](#) Method

Syntax: Visual Basic

```
NotOverridable Public Sub Commit() _  
    Implements IDbTransaction.Commit
```

Syntax: C#

```
public void Commit();
```

Implements

IDbTransaction.Commit

See Also

[MySQLTransaction Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.10.1.1.1.4. [MySQLTransaction.Rollback](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Sub Rollback() _  
    Implements IDbTransaction.Rollback
```

Syntax: C#

```
public void Rollback();
```

Implements

IDbTransaction.Rollback

See Also

[MySQLTransaction Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.10.2. [MySQLConnection.BeginTransaction](#) Method**Syntax: Visual Basic**

```
Overloads Public Function BeginTransaction( _  
    ByVal iso As IsolationLevel _  
) As MySQLTransaction
```

Syntax: C#

```
public MySQLTransaction BeginTransaction(  
    IsolationLevel iso  
);
```

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLConnection.BeginTransaction Overload List](#)

26.2.4.1.2.1.1.3.1.1.11. [MySQLConnection.ChangeDatabase](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Sub ChangeDatabase( _  
    ByVal databaseName As String _  
) _  
    Implements IDbConnection.ChangeDatabase
```

Syntax: C#

```
public void ChangeDatabase(  
    string databaseName  
);
```

Implements

IDbConnection.ChangeDatabase

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.12. [MySQLConnection.Close](#) Method

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _  
    Implements IDbConnection.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDbConnection.Close

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.13. [MySQLConnection.CreateCommand](#) Method

Syntax: Visual Basic

```
Public Function CreateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand CreateCommand();
```

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.14. [MySQLConnection.Open](#) Method

Syntax: Visual Basic

```
NotOverridable Public Sub Open() _  
    Implements IDbConnection.Open
```

Syntax: C#

```
public void Open();
```

Implements

IDbConnection.Open

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.15. [MySQLConnection.Ping](#) Method

Ping

Syntax: Visual Basic

```
Public Function Ping() As Boolean
```

Syntax: C#


```
public bool Ping();
```

Return Value**See Also**

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16. [MySqlConnection.InfoMessage](#) Event

Syntax: Visual Basic

```
Public Event InfoMessage As MySqlInfoMessageEventHandler
```

Syntax: C#

```
public event MySqlInfoMessageEventHandler InfoMessage;
```

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1. [MySqlInfoMessageEventHandler](#) Delegate

Represents the method that will handle the [InfoMessage](#) event of a [MySqlConnection](#) .

Syntax: Visual Basic

```
Public Delegate Sub MySqlInfoMessageEventHandler( _  
    ByVal sender As Object, _  
    ByVal args As MySqlInfoMessageEventArgs _  
)
```

Syntax: C#

```
public delegate void MySqlInfoMessageEventHandler(  
    object sender,  
    MySqlInfoMessageEventArgs args  
);
```

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1.1. [MySqlInfoMessageEventArgs](#) Class

Provides data for the InfoMessage event. This class cannot be inherited.

For a list of all members of this type, see [MySqlInfoMessageEventArgs Members](#) .

Syntax: Visual Basic

```
Public Class MySqlInfoMessageEventArgs_  
    Inherits EventArgs
```

Syntax: C#

```
public class MySqlInfoMessageEventArgs : EventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlInfoMessageEventArgs Members](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1.1.1.1. [MySqlInfoMessageEventArgs](#) Members

[MySqlInfoMessageEventArgs](#) overview

Public Instance Constructors

MySqlInfoMessageEventArgs Constructor	Initializes a new instance of the MySqlInfoMessageEventArgs class.

Public Instance Fields

errors	

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Type of the current instance.
ToString(inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize(inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySqlInfoMessageEventArgs](#) Class , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1.1.1.1.1. [MySqlInfoMessageEventArgs](#) Constructor

Initializes a new instance of the [MySqlInfoMessageEventArgs](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlInfoMessageEventArgs();
```

See Also

[MySqlInfoMessageEventArgs Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1.1.1.2. [MySqlInfoMessageEventArgs.errors](#) Field**Syntax: Visual Basic**

```
Public errors As MySqlError()
```

Syntax: C#

```
public MySqlError[] errors;
```

See Also

[MySqlInfoMessageEventArgs Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1. [MySqlError](#) Class

Collection of error codes that can be returned by the server

For a list of all members of this type, see [MySqlError Members](#) .

Syntax: Visual Basic

```
Public Class MySqlError
```

Syntax: C#

```
public class MySqlError
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlError Members](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1.1. [MySqlError](#) Members

[MySqlError overview](#)

Public Instance Constructors

MySqlError Constructor	

Public Instance Properties

Code	Error code
Level	Error level
Message	Error message

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Type of the current instance.
ToString(inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize(inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySqlError Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1.1.1. [MySqlError](#) Constructor**Syntax: Visual Basic**

```
Public Sub New( _
    ByVal level As String, _
    ByVal code As Integer, _
    ByVal message As String _
)
```

Syntax: C#

```
public MySqlError(
    string level,
    int code,
    string message
);
```

Parameters

- **level:**
- **code:**
- **message:**

See Also

[MySqlError Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1.1.2. Code Property

Error code

Syntax: Visual Basic

```
Public ReadOnly Property Code As Integer
```

Syntax: C#

```
public int Code {get;}
```

See Also

[MySqlError Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1.1.3. Level Property

Error level

Syntax: Visual Basic

```
Public ReadOnly Property Level As String
```

Syntax: C#

```
public string Level {get;}
```

See Also

[MySqlError Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1.1.4. Message Property

Error message

Syntax: Visual Basic

```
Public ReadOnly Property Message As String
```

Syntax: C#

```
public string Message {get;}
```

See Also

[MySqlError Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.3.1.1.17. [MySqlConnection.StateChange](#) Event

Syntax: Visual Basic

```
Public Event StateChange As StateChangeEventHandler
```

Syntax: C#

```
public event StateChangeEventHandler StateChange;
```

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.1.4. [MySqlCommand](#) Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String, _  
    ByVal connection As MySqlConnection, _  
    ByVal transaction As MySqlTransaction _  
)
```

Syntax: C#

```
public MySqlCommand(  
    string cmdText,  
    MySqlConnection connection,  
    MySqlTransaction transaction  
)
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlCommand Constructor Overload List](#)

26.2.4.1.2.1.2. CommandText Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandText As String _  
    Implements IDbCommand.CommandText
```

Syntax: C#

```
public string CommandText {get; set;}
```

Implements

IDbCommand.CommandText

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.3. CommandTimeout Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandTimeout As Integer _  
    Implements IDbCommand.CommandTimeout
```

Syntax: C#

```
public int CommandTimeout {get; set;}
```

Implements

IDbCommand.CommandTimeout

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.4. CommandType Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandType As CommandType _  
    Implements IDbCommand.CommandType
```

Syntax: C#

```
public System.Data.CommandType CommandType {get; set;}
```

Implements

IDbCommand.CommandType

See Also

[MySqlCommand Class](#) , [MySqlConnection Namespace](#)

26.2.4.1.2.1.5. Connection Property

Syntax: Visual Basic

```
Public Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get; set;}
```

See Also

[MySqlCommand Class](#) , [MySqlConnection Namespace](#)

26.2.4.1.2.1.6. IsPrepared Property

Syntax: Visual Basic

```
Public ReadOnly Property IsPrepared As Boolean
```

Syntax: C#

```
public bool IsPrepared {get;}
```

See Also

[MySqlCommand Class](#) , [MySqlConnection Namespace](#)

26.2.4.1.2.1.7. Parameters Property

Syntax: Visual Basic

```
Public ReadOnly Property Parameters As MySqlParameterCollection
```

Syntax: C#

```
public MySqlParameterCollection Parameters {get;}
```

See Also

[MySqlCommand Class](#) , [MySqlConnection Namespace](#)

26.2.4.1.2.1.7.1. [MySqlParameterCollection](#) Class

Represents a collection of parameters relevant to a [MySqlCommand](#) as well as their respective mappings to columns in a DataSet. This class cannot be inherited.

For a list of all members of this type, see [MySqlParameterCollection Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnection_  
    Inherits MarshalByRefObject_  
    Implements IDataParameterCollection, IList, ICollection, IEnumerable
```

Syntax: C#

```
public sealed class MySqlConnection : MarshalByRefObject, IDataParameterCollection, IList, ICollection, IEnumerable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLParameterCollection Members](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1. MySQLParameterCollection Members

[MySQLParameterCollection overview](#)

Public Instance Constructors

MySQLParameterCollection Constructor	Initializes a new instance of the MySQLParameterCollection class.

Public Instance Properties

Count	Gets the number of MySQLParameter objects in the collection.
Item	Overloaded. Gets the MySQLParameter with a specified attribute. In C#, this property is the indexer for the MySQLParameterCollection class.

Public Instance Methods

Add	Overloaded. Adds the specified MySQLParameter object to the MySQLParameterCollection .
Clear	Removes all items from the collection.
Contains	Overloaded. Gets a value indicating whether a MySQLParameter exists in the collection.
CopyTo	Copies MySQLParameter objects from the MySQLParameterCollection to the specified array.
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Type of the current instance.
IndexOf	Overloaded. Gets the location of a MySQLParameter in the collection.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Insert	Inserts a MySQLParameter into the collection at the specified index.
Remove	Removes the specified MySQLParameter from the collection.

RemoveAt	Overloaded. Removes the specified MySQLParameter from the collection.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.1. [MySQLParameterCollection](#) Constructor

Initializes a new instance of the [MySQLParameterCollection](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySQLParameterCollection();
```

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.2. Count Property

Gets the number of [MySQLParameter](#) objects in the collection.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Count As Integer _  
    Implements ICollection.Count
```

Syntax: C#

```
public int Count {get;}
```

Implements

ICollection.Count

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3. Item Property

Gets the [MySQLParameter](#) with a specified attribute. In C#, this property is the indexer for the [MySQLParameterCollection](#) class.

Overload List

Gets the [MySQLParameter](#) at the specified index.

- [public MySQLParameter this\[int\] {get; set;}](#)

Gets the [MySQLParameter](#) with the specified name.

- [public MySQLParameter this\[string\] {get; set;}](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1. [MySQLParameter](#) Class

Represents a parameter to a [MySQLCommand](#) , and optionally, its mapping to DataSetcolumns. This class cannot be inherited.

For a list of all members of this type, see [MySQLParameter Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySQLParameter_
    Inherits MarshalByRefObject_
    Implements IDataParameter, IDbDataParameter, ICloneable
```

Syntax: C#

```
public sealed class MySQLParameter : MarshalByRefObject, IDataParameter, IDbDataParameter, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLParameter Members](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1. [MySQLParameter](#) Members

[MySQLParameter overview](#)

Public Instance Constructors

MySQLParameter	Overloaded. Initializes a new instance of the MySQLParameter class.

Public Instance Properties

DbType	Gets or sets the DbType of the parameter.
Direction	Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. As of MySQL version 4.1 and earlier, input-only is the only valid choice.
IsNullable	Gets or sets a value indicating whether the parameter accepts null values.
IsUnsigned	
MySQLDbType	Gets or sets the MySQLDbType of the parameter.
ParameterName	Gets or sets the name of the MySQLParameter.
Precision	Gets or sets the maximum number of digits used to represent the Value property.
Scale	Gets or sets the number of decimal places to which Value is resolved.

Size	Gets or sets the maximum size, in bytes, of the data within the column.
SourceColumn	Gets or sets the name of the source column that is mapped to the DataSet and used for loading or returning the Value .
SourceVersion	Gets or sets the DataRowVersion to use when loading Value .
Value	Gets or sets the value of the parameter.

Public Instance Methods

CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ToString	Overridden. Gets a string containing the ParameterName .

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.1. [MySQLParameter](#) Constructor

Initializes a new instance of the MySQLParameter class.

Overload List

Initializes a new instance of the MySQLParameter class.

- [public MySQLParameter\(\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name and the data type.

- [public MySQLParameter\(string, MySQLDbType\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySQLDbType](#) , and the size.

- [public MySQLParameter\(string, MySQLDbType, int\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the type of the parameter, the size of the parameter, a ParameterDirection, the precision of the parameter, the scale of the parameter, the source column, a DataRowVersion to use, and the value of the parameter.

- [public MySQLParameter\(string, MySQLDbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySQLDbType](#) , the size, and the source column name.

- [public MySQLParameter\(string,MySQLDbType,int,string\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name and a value of the new MySQLParameter.

- [public MySQLParameter\(string,object\);](#)

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.1.1. [MySQLParameter](#) Constructor ()

Initializes a new instance of the MySQLParameter class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySQLParameter();
```

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameter Constructor Overload List](#)

26.2.4.1.2.1.7.1.1.3.1.1.1.2. [MySQLParameter](#) Constructor

Initializes a new instance of the [MySQLParameter](#) class with the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySQLDbType _  
)
```

Syntax: C#

```
public MySQLParameter(  
    stringparameterName,  
    MySQLDbTypedbType  
);
```

Parameters

- [parameterName](#): The name of the parameter to map.
- [dbType](#): One of the [MySQLDbType](#) values.

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameter Constructor Overload List](#)

26.2.4.1.2.1.7.1.1.3.1.1.1.2.1. [MySQLDbType](#) Enumeration

Specifies MySQL specific data type of a field, property, for use in a [MySQLParameter](#) .

Syntax: Visual Basic

```
Public Enum MySqlDbType
```

Syntax: C#

```
public enum MySqlDbType
```

Members

Member Name	Description
VarChar	A variable-length string containing 0 to 65535 characters
Timestamp	A timestamp. The range is '1970-01-01 00:00:01' to sometime in the year 2038
LongBlob	A BLOB or TEXT column with a maximum length of 4294967295 or 4G (2 ³² - 1) characters
Time	The range is '-838:59:59' to '838:59:59'.
TinyBlob	A BLOB or TEXT column with a maximum length of 255 (2 ⁸ - 1) characters
Datetime	The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
Decimal	A fixed precision and scale numeric value between -1038 -1 and 10 38 -1.
UByte	
Blob	A BLOB or TEXT column with a maximum length of 65535 (2 ¹⁶ - 1) characters
Double	A normal-size (double-precision) floating-point number. Allowable values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308.
Newdate	Obsolete Use Datetime or Date type
Byte	The signed range is -128 to 127. The unsigned range is 0 to 255.
Date	Date The supported range is '1000-01-01' to '9999-12-31'.
VarChar	A variable-length string containing 0 to 255 characters
UInt16	
UInt24	
Int16	A 16-bit signed integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535
NewDecimal	New Decimal
Set	A set. A string object that can have zero or more values, each of which must be chosen from the list of values 'value1', 'value2', ... A SET can have a maximum of 64 members.
String	Obsolete Use VarChar type
Enum	An enumeration. A string object that can have only one value, chosen from the list of values 'value1', 'value2', ..., NULL or the special "" error value. An ENUM can have a maximum of 65535 distinct values
Geometry	
UInt64	

Int64	A 64-bit signed integer.
UInt32	
Int24	Specifies a 24 (3 byte) signed or unsigned value.
Bit	Bit-field data type
Float	A small (single-precision) floating-point number. Allowable values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38.
Year	A year in 2- or 4-digit format (default is 4-digit). The allowable values are 1901 to 2155, 0000 in the 4-digit year format, and 1970-2069 if you use the 2-digit format (70-69)
Int32	A 32-bit signed integer
MediumBlob	A BLOB or TEXT column with a maximum length of 16777215 (2 ²⁴ - 1) characters

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.3. [MySQLParameter](#) Constructor (String, MySqlDbType, Int32)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySqlDbType](#), and the size.

Syntax: Visual Basic

```
Overloads Public Sub New(
    ByVal parameterName As String, _
    ByVal dbType As MySqlDbType, _
    ByVal size As Integer _
)
```

Syntax: C#

```
public MySQLParameter(
    string parameterName,
    MySqlDbType dbType,
    int size
);
```

Parameters

- **parameterName**: The name of the parameter to map.
- **dbType**: One of the [MySqlDbType](#) values.
- **size**: The length of the parameter.

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLParameter Constructor Overload List](#)

26.2.4.1.2.1.7.1.1.3.1.1.4. [MySQLParameter](#) Constructor

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the type of the parameter, the size of the parameter, a [ParameterDirection](#), the precision of the parameter, the scale of the parameter, the source column, a [DataRowVersion](#) to use, and the value of the parameter.

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal parameterName As String, _
    ByVal dbType As MySqlDbType, _
    ByVal size As Integer, _
    ByVal direction As ParameterDirection, _
    ByVal isNullable As Boolean, _
    ByVal precision As Byte, _
    ByVal scale As Byte, _
    ByVal sourceColumn As String, _
    ByVal sourceVersion As DataRowVersion, _
    ByVal value As Object _
)
```

Syntax: C#

```
public MySQLParameter(
    string parameterName,
    MySqlDbType dbType,
    int size,
    ParameterDirection direction,
    bool isNullable,
    byte precision,
    byte scale,
    string sourceColumn,
    DataRowVersion sourceVersion,
    object value
);
```

Parameters

- **parameterName**: The name of the parameter to map.
- **dbType**: One of the [MySQLDbType](#) values.
- **size**: The length of the parameter.
- **direction**: One of the [ParameterDirection](#) values.
- **isNullable**: true if the value of the field can be null, otherwise false.
- **precision**: The total number of digits to the left and right of the decimal point to which [Value](#) is resolved.
- **scale**: The total number of decimal places to which [Value](#) is resolved.
- **sourceColumn**: The name of the source column.
- **sourceVersion**: One of the [DataRowVersion](#) values.
- **value**: An [Object](#) that is the value of the [MySQLParameter](#).

Exceptions

Exception Type	Condition
ArgumentException	

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLParameter Constructor Overload List](#)

26.2.4.1.2.1.7.1.1.3.1.1.1.4.1. Value Property

Gets or sets the value of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property Value As Object _  
    Implements IDataParameter.Value
```

Syntax: C#

```
public object Value {get; set;}
```

Implements

IDataParameter.Value

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.5. [MySQLParameter](#) Constructor

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySQLDbType](#) , the size, and the source column name.

Syntax: Visual Basic

```
Overloads Public Sub New(  
    ByVal parameterName As String, _  
    ByVal dbType As MySQLDbType, _  
    ByVal size As Integer, _  
    ByVal sourceColumn As String _  
)
```

Syntax: C#

```
public MySQLParameter(  
    stringparameterName,  
    MySQLDbTypedbType,  
    intsize,  
    stringsourceColumn  
);
```

Parameters

- [parameterName](#): The name of the parameter to map.
- [dbType](#): One of the [MySQLDbType](#) values.
- [size](#): The length of the parameter.
- [sourceColumn](#): The name of the source column.

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameter Constructor Overload List](#)

26.2.4.1.2.1.7.1.1.3.1.1.6. [MySQLParameter](#) Constructor

Initializes a new instance of the [MySQLParameter](#) class with the parameter name and a value of the new MySQLParameter.

Syntax: Visual Basic

```
Overloads Public Sub New(  
    ByVal parameterName As String, _  
    ByVal value As Object _  
)
```


Syntax: C#

```
public MySqlParameter(  
    string parameterName,  
    object value  
)
```

Parameters

- **parameterName**: The name of the parameter to map.
- **value**: An Object that is the value of the [MySqlParameter](#).

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

26.2.4.1.2.1.7.1.1.3.1.1.2. DbType Property

Gets or sets the DbType of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property DbType As DbType _  
    Implements IDataParameter.DbType
```

Syntax: C#

```
public System.Data.DbType DbType {get; set;}
```

Implements

IDataParameter.DbType

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.3. Direction Property

Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. As of MySQL version 4.1 and earlier, input-only is the only valid choice.

Syntax: Visual Basic

```
NotOverridable Public Property Direction As ParameterDirection _  
    Implements IDataParameter.Direction
```

Syntax: C#

```
public System.Data.ParameterDirection Direction {get; set;}
```

Implements

IDataParameter.Direction

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.4. IsNullable Property

Gets or sets a value indicating whether the parameter accepts null values.

Syntax: Visual Basic

```
NotOverridable Public Property IsNullable As Boolean _  
    Implements IDataParameter.IsNullable
```

Syntax: C#

```
public bool IsNullable {get; set;}
```

Implements

IDataParameter.IsNullable

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.5. IsUnsigned Property

Syntax: Visual Basic

```
Public Property IsUnsigned As Boolean
```

Syntax: C#

```
public bool IsUnsigned {get; set;}
```

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.6. MySQLDbType Property

Gets or sets the MySQLDbType of the parameter.

Syntax: Visual Basic

```
Public Property MySQLDbType As MySQLDbType
```

Syntax: C#

```
public MySQLDbType MySQLDbType {get; set;}
```

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.7. ParameterName Property

Gets or sets the name of the MySQLParameter.

Syntax: Visual Basic

```
NotOverridable Public Property ParameterName As String _  
    Implements IDataParameter.ParameterName
```

Syntax: C#

```
public string ParameterName {get; set;}
```

Implements

IDataParameter.ParameterName

See Also[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.8. Precision Property

Gets or sets the maximum number of digits used to represent the [Value](#) property.

Syntax: Visual Basic

```
NotOverridable Public Property Precision As Byte _  
    Implements IDbDataParameter.Precision
```

Syntax: C#

```
public byte Precision {get; set;}
```

Implements

IDbDataParameter.Precision

See Also[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.9. Scale Property

Gets or sets the number of decimal places to which [Value](#) is resolved.

Syntax: Visual Basic

```
NotOverridable Public Property Scale As Byte _  
    Implements IDbDataParameter.Scale
```

Syntax: C#

```
public byte Scale {get; set;}
```

Implements

IDbDataParameter.Scale

See Also[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.10. Size Property

Gets or sets the maximum size, in bytes, of the data within the column.

Syntax: Visual Basic

```
NotOverridable Public Property Size As Integer _  
    Implements IDbDataParameter.Size
```

Syntax: C#

```
public int Size {get; set;}
```

Implements

IDbDataParameter.Size

See Also[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.11. SourceColumn Property

Gets or sets the name of the source column that is mapped to the DataSet and used for loading or returning the [Value](#) .

Syntax: Visual Basic

```
NotOverridable Public Property SourceColumn As String _  
    Implements IDataParameter.SourceColumn
```

Syntax: C#

```
public string SourceColumn {get; set;}
```

Implements

IDataParameter.SourceColumn

See Also[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.1.1.12. SourceVersion Property

Gets or sets the DataRowVersion to use when loading [Value](#) .

Syntax: Visual Basic

```
NotOverridable Public Property SourceVersion As DataRowVersion _  
    Implements IDataParameter.SourceVersion
```

Syntax: C#

```
public System.Data.DataRowVersion SourceVersion {get; set;}
```

Implements

IDataParameter.SourceVersion

See Also[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)26.2.4.1.2.1.7.1.1.3.1.1.13. [MySQLParameter.ToString](#) Method

Overridden. Gets a string containing the [ParameterName](#) .

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

Return Value

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.3.2. Item Property (Int32)

Gets the [MySQLParameter](#) at the specified index.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _  
    ByVal index As Integer _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter this[  
    int index  
] {get; set;}
```

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.Item Overload List](#)

26.2.4.1.2.1.7.1.1.3.3. Item Property (String)

Gets the [MySQLParameter](#) with the specified name.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _  
    ByVal name As String _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter this[  
    string name  
] {get; set;}
```

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.Item Overload List](#)

26.2.4.1.2.1.7.1.1.4. Add Method

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#) .

Overload List

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#) .

- [public MySQLParameter Add\(MySQLParameter\);](#)

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#) .

- [public int Add\(object\);](#)

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the parameter name and the data type.

- [public MySQLParameter Add\(string,MySQLDbType\);](#)

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) with the parameter name, the data type, and the column length.

- `public MySQLParameter Add(string,MySQLDbType,int);`

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) with the parameter name, the data type, the column length, and the source column name.

- `public MySQLParameter Add(string,MySQLDbType,int,string);`

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the specified parameter name and value.

- `public MySQLParameter Add(string,object);`

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.4.1. [MySQLParameterCollection.Add](#) Method

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#) .

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal value As MySQLParameter _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter Add(  
    MySQLParametervalue  
);
```

Parameters

- **value**: The [MySQLParameter](#) to add to the collection.

Return Value

The newly added [MySQLParameter](#) object.

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.Add Overload List](#)

26.2.4.1.2.1.7.1.1.4.2. [MySQLParameterCollection.Add](#) Method

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#) .

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Add( _  
    ByVal value As Object _  
) As Integer _  
    Implements IList.Add
```

Syntax: C#

```
public int Add(  

```

```
objectvalue  
);
```

Parameters

- **value**: The [MySQLParameter](#) to add to the collection.

Return Value

The index of the new [MySQLParameter](#) object.

Implements

[IList.Add](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.Add Overload List](#)

26.2.4.1.2.1.7.1.1.4.3. [MySQLParameterCollection.Add](#) Method

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter Add(  
    stringparameterName,  
    MySqlDbTypedbType  
);
```

Parameters

- **parameterName**: The name of the parameter.
- **dbType**: One of the [MySQLDbType](#) values.

Return Value

The newly added [MySQLParameter](#) object.

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.Add Overload List](#)

26.2.4.1.2.1.7.1.1.4.4. [MySQLParameterCollection.Add](#) Method

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) with the parameter name, the data type, and the column length.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType, _  
    ByVal size As Integer _  
) As MySQLParameter
```

Syntax: C#

```
public MySqlParameter Add(  
    string parameterName,  
    MySqlDbType dbType,  
    int size  
);
```

Parameters

- **parameterName**: The name of the parameter.
- **dbType**: One of the [MySqlDbType](#) values.
- **size**: The length of the column.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySqlParameterCollection Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlParameterCollection.Add Overload List](#)

26.2.4.1.2.1.7.1.1.4.5. [MySqlParameterCollection.Add](#) Method

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) with the parameter name, the data type, the column length, and the source column name.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType, _  
    ByVal size As Integer, _  
    ByVal sourceColumn As String _  
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(  
    string parameterName,  
    MySqlDbType dbType,  
    int size,  
    string sourceColumn  
);
```

Parameters

- **parameterName**: The name of the parameter.
- **dbType**: One of the [MySqlDbType](#) values.
- **size**: The length of the column.
- **sourceColumn**: The name of the source column.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySqlParameterCollection Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlParameterCollection.Add Overload List](#)

26.2.4.1.2.1.7.1.1.4.6. [MySqlParameterCollection.Add](#) Method

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the specified parameter name and value.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal value As Object _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter Add(  
    string parameterName,  
    object value  
);
```

Parameters

- **parameterName**: The name of the parameter.
- **value**: The **Value** of the [MySQLParameter](#) to add to the collection.

Return Value

The newly added [MySQLParameter](#) object.

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.Add Overload List](#)

26.2.4.1.2.1.7.1.1.5. [MySQLParameterCollection.Clear](#) Method

Removes all items from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Clear() _  
    Implements IList.Clear
```

Syntax: C#

```
public void Clear();
```

Implements

IList.Clear

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.6. Contains Method

Gets a value indicating whether a [MySQLParameter](#) exists in the collection.

Overload List

Gets a value indicating whether a [MySQLParameter](#) exists in the collection.

- **public bool Contains(object);**

Gets a value indicating whether a [MySQLParameter](#) with the specified parameter name exists in the collection.

- `public bool Contains(string);`

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.6.1. [MySQLParameterCollection.Contains](#) Method

Gets a value indicating whether a [MySQLParameter](#) exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _  
    ByVal value As Object _  
) As Boolean _  
    Implements IList.Contains
```

Syntax: C#

```
public bool Contains(  
    objectvalue  
);
```

Parameters

- **value**: The value of the [MySQLParameter](#) object to find.

Return Value

true if the collection contains the [MySQLParameter](#) object; otherwise, false.

Implements

[IList.Contains](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.Contains Overload List](#)

26.2.4.1.2.1.7.1.1.6.2. [MySQLParameterCollection.Contains](#) Method

Gets a value indicating whether a [MySQLParameter](#) with the specified parameter name exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _  
    ByVal name As String _  
) As Boolean _  
    Implements IDataParameterCollection.Contains
```

Syntax: C#

```
public bool Contains(  
    stringname  
);
```

Parameters

- **name**: The name of the [MySQLParameter](#) object to find.

Return Value

true if the collection contains the parameter; otherwise, false.

Implements

IDataParameterCollection.Contains

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.Contains Overload List](#)

26.2.4.1.2.1.7.1.1.7. [MySQLParameterCollection.CopyTo](#) Method

Copies MySQLParameter objects from the MySQLParameterCollection to the specified array.

Syntax: Visual Basic

```
NotOverridable Public Sub CopyTo( _  
    ByVal array As Array, _  
    ByVal index As Integer _  
) _  
    Implements ICollection.CopyTo
```

Syntax: C#

```
public void CopyTo(  
    Arrayarray,  
    intindex  
);
```

Parameters

- [array](#):
- [index](#):

Implements

ICollection.CopyTo

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.8. [IndexOf](#) Method

Gets the location of a [MySQLParameter](#) in the collection.

Overload List

Gets the location of a [MySQLParameter](#) in the collection.

- [public int IndexOf\(object\);](#)

Gets the location of the [MySQLParameter](#) in the collection with a specific parameter name.

- [public int IndexOf\(string\);](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.8.1. [MySQLParameterCollection.IndexOf](#) Method

Gets the location of a [MySQLParameter](#) in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _  
    ByVal value As Object _  
) As Integer _  
    Implements IList.IndexOf
```

Syntax: C#

```
public int IndexOf(  
    objectvalue  
);
```

Parameters

- **value**: The [MySQLParameter](#) object to locate.

Return Value

The zero-based location of the [MySQLParameter](#) in the collection.

Implements

[IList.IndexOf](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.IndexOf Overload List](#)

26.2.4.1.2.1.7.1.1.8.2. [MySQLParameterCollection.IndexOf](#) Method

Gets the location of the [MySQLParameter](#) in the collection with a specific parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _  
    ByVal parameterName As String _  
) As Integer _  
    Implements IDataParameterCollection.IndexOf
```

Syntax: C#

```
public int IndexOf(  
    stringparameterName  
);
```

Parameters

- **parameterName**: The name of the [MySQLParameter](#) object to retrieve.

Return Value

The zero-based location of the [MySQLParameter](#) in the collection.

Implements

[IDataParameterCollection.IndexOf](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.IndexOf Overload List](#)

26.2.4.1.2.1.7.1.1.9. [MySQLParameterCollection.Insert](#) Method

Inserts a `MySQLParameter` into the collection at the specified index.

Syntax: Visual Basic

```
NotOverridable Public Sub Insert( _  
    ByVal index As Integer, _  
    ByVal value As Object _  
) _  
    Implements IList.Insert
```

Syntax: C#

```
public void Insert(  
    int index,  
    object value  
);
```

Parameters

- `index`:
- `value`:

Implements

`IList.Insert`

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.10. [MySQLParameterCollection.Remove](#) Method

Removes the specified `MySQLParameter` from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Remove( _  
    ByVal value As Object _  
) _  
    Implements IList.Remove
```

Syntax: C#

```
public void Remove(  
    object value  
);
```

Parameters

- `value`:

Implements

`IList.Remove`

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.11. RemoveAt Method

Removes the specified [MySQLParameter](#) from the collection.

Overload List

Removes the specified [MySQLParameter](#) from the collection using a specific index.

- [public void RemoveAt\(int\);](#)

Removes the specified [MySQLParameter](#) from the collection using the parameter name.

- [public void RemoveAt\(string\);](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.7.1.1.11.1. MySQLParameterCollection.RemoveAt Method

Removes the specified [MySQLParameter](#) from the collection using a specific index.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _  
    ByVal index As Integer _  
) _  
    Implements IList.RemoveAt
```

Syntax: C#

```
public void RemoveAt(  
    int index  
);
```

Parameters

- [index](#): The zero-based index of the parameter.

Implements

IList.RemoveAt

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.RemoveAt Overload List](#)

26.2.4.1.2.1.7.1.1.11.2. MySQLParameterCollection.RemoveAt Method

Removes the specified [MySQLParameter](#) from the collection using the parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _  
    ByVal name As String _  
) _  
    Implements IDataParameterCollection.RemoveAt
```

Syntax: C#

```
public void RemoveAt(  
    stringname  
);
```

Parameters

- **name**: The name of the [MySQLParameter](#) object to retrieve.

Implements

IDataParameterCollection.RemoveAt

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameterCollection.RemoveAt Overload List](#)

26.2.4.1.2.1.8. Transaction Property

Syntax: Visual Basic

```
Public Property Transaction As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Transaction {get; set;}
```

See Also

[MySQLCommand Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.9. UpdatedRowSource Property

Syntax: Visual Basic

```
NotOverridable Public Property UpdatedRowSource As UpdateRowSource _  
    Implements IDbCommand.UpdatedRowSource
```

Syntax: C#

```
public System.Data.UpdateRowSource UpdatedRowSource {get; set;}
```

Implements

IDbCommand.UpdatedRowSource

See Also

[MySQLCommand Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.10. [MySQLCommand.Cancel](#) Method

Attempts to cancel the execution of a MySqlCommand. This operation is not supported.

Syntax: Visual Basic

```
NotOverridable Public Sub Cancel() _  
    Implements IDbCommand.Cancel
```

Syntax: C#

```
public void Cancel();
```

Implements

IDbCommand.Cancel

Remarks

Cancelling an executing command is currently not supported on any version of MySQL.

Exceptions

Exception Type	Condition
NotSupportedException	This operation is not supported.

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.11. MySqlCommand.CreateParameter Method

Creates a new instance of a [MySqlParameter](#) object.

Syntax: Visual Basic

```
Public Function CreateParameter() As MySqlParameter
```

Syntax: C#

```
public MySqlParameter CreateParameter();
```

Return Value

A [MySqlParameter](#) object.

Remarks

This method is a strongly-typed version of CreateParameter.

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.12. MySqlCommand.ExecuteNonQuery Method

Syntax: Visual Basic

```
NotOverridable Public Function ExecuteNonQuery() As Integer _  
    Implements IDbCommand.ExecuteNonQuery
```

Syntax: C#

```
public int ExecuteNonQuery();
```

Implements

IDbCommand.ExecuteNonQuery

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13. ExecuteReader Method

Overload List

- [public MySqlDataReader ExecuteReader\(\);](#)
- [public MySqlDataReader ExecuteReader\(CommandBehavior\);](#)

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1. MySqlCommand.ExecuteReader Method

Syntax: Visual Basic

```
Overloads Public Function ExecuteReader() As MySqlDataReader
```

Syntax: C#

```
public MySqlDataReader ExecuteReader();
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlCommand.ExecuteReader Overload List](#)

26.2.4.1.2.1.13.1.1. MySqlDataReader Class

Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.

For a list of all members of this type, see [MySqlDataReader Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataReader_  
    Inherits MarshalByRefObject_  
    Implements IEnumerable, IDataReader, IDisposable, IDataRecord
```

Syntax: C#

```
public sealed class MySqlDataReader : MarshalByRefObject, IEnumerable, IDataReader, IDisposable, IDataRecord
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlDataReader Members](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1. MySqlDataReader Members

[MySqlDataReader overview](#)

Public Instance Properties

Depth	Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.
FieldCount	Gets the number of columns in the current row.
HasRows	Gets a value indicating whether the MySQLDataReader contains one or more rows.
IsClosed	Gets a value indicating whether the data reader is closed.
Item	Overloaded. Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySQLDataReader class.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Public Instance Methods

Close	Closes the MySQLDataReader object.
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBoolean	Gets the value of the specified column as a Boolean.
GetByte	Gets the value of the specified column as a byte.
GetBytes	Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.
GetChar	Gets the value of the specified column as a single character.
GetChars	Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.
GetDataTypeName	Gets the name of the source data type.
GetDateTime	
GetDecimal	
GetDouble	
GetFieldType	Gets the Type that is the data type of the object.
GetFloat	
GetGuid	
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetInt16	
GetInt32	
GetInt64	
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetMySQLDateTime	
GetName	Gets the name of the specified column.
GetOrdinal	Gets the column ordinal, given the name of the column.
GetSchemaTable	Returns a DataTable that describes the column metadata of the MySQLDataReader.
GetString	
GetTimeSpan	

GetType(inherited from Object)	Gets the Type of the current instance.
GetUInt16	
GetUInt32	
GetUInt64	
GetValue	Gets the value of the specified column in its native format.
GetValues	Gets all attribute columns in the collection for the current row.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
IsDBNull	Gets a value indicating whether the column contains non-existent or missing values.
NextResult	Advances the data reader to the next result, when reading the results of batch SQL statements.
Read	Advances the MySQLDataReader to the next record.
ToString(inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.1. Depth Property

Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Depth As Integer _
    Implements IDataReader.Depth
```

Syntax: C#

```
public int Depth {get;}
```

Implements

IDataReader.Depth

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.2. FieldCount Property

Gets the number of columns in the current row.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property FieldCount As Integer _
    Implements IDataRecord.FieldCount
```

Syntax: C#

```
public int FieldCount {get;}
```

Implements

IDataRecord.FieldCount

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.3. HasRows Property

Gets a value indicating whether the MySQLDataReader contains one or more rows.

Syntax: Visual Basic

```
Public ReadOnly Property HasRows As Boolean
```

Syntax: C#

```
public bool HasRows {get;}
```

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.4. IsClosed Property

Gets a value indicating whether the data reader is closed.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsClosed As Boolean _  
    Implements IDataReader.IsClosed
```

Syntax: C#

```
public bool IsClosed {get;}
```

Implements

IDataReader.IsClosed

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.5. Item Property

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySQLDataReader class.

Overload List

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySQLDataReader class.

- [public object this\[int\] {get;}](#)

Gets the value of a column in its native format. In C#, this property is the indexer for the MySQLDataReader class.

- [public object this\[string\] {get;}](#)

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.5.1. Item Property (Int32)

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _  
    ByVal i As Integer _  
) _  
- Implements IDataRecord.Item As Object _  
- Implements IDataRecord.Item
```

Syntax: C#

```
public object this[  
    inti  
] {get;}
```

Implements

`IDataRecord.Item`

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlDataReader.Item Overload List](#)

26.2.4.1.2.1.13.1.1.1.5.2. Item Property (String)

Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _  
    ByVal name As String _  
) _  
- Implements IDataRecord.Item As Object _  
- Implements IDataRecord.Item
```

Syntax: C#

```
public object this[  
    stringname  
] {get;}
```

Implements

`IDataRecord.Item`

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlDataReader.Item Overload List](#)

26.2.4.1.2.1.13.1.1.1.6. RecordsAffected Property

Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property RecordsAffected As Integer _  
- Implements IDataReader.RecordsAffected
```

Syntax: C#

```
public int RecordsAffected {get;}
```

Implements

IDataReader.RecordsAffected

See Also[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)**26.2.4.1.2.1.13.1.1.7. [MySqlDataReader.Close](#) Method**

Closes the MySqlDataReader object.

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _  
    Implements IDataReader.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDataReader.Close

See Also[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)**26.2.4.1.2.1.13.1.1.8. [MySqlDataReader.GetBoolean](#) Method**

Gets the value of the specified column as a Boolean.

Syntax: Visual Basic

```
NotOverridable Public Function GetBoolean( _  
    ByVal i As Integer _  
) As Boolean _  
    Implements IDataRecord.GetBoolean
```

Syntax: C#

```
public bool GetBoolean(  
    inti  
);
```

Parameters

- [i](#):

Return Value**Implements**

IDataRecord.GetBoolean

See Also[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)**26.2.4.1.2.1.13.1.1.9. [MySqlDataReader.GetByte](#) Method**

Gets the value of the specified column as a byte.

Syntax: Visual Basic

```
NotOverridable Public Function GetByte( _  
    ByVal i As Integer _  
    ) As Byte _  
    Implements IDataRecord.GetByte
```

Syntax: C#

```
public byte GetByte(  
    inti  
);
```

Parameters

- **i**:

Return Value**Implements**

IDataRecord.GetByte

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.10. [MySQLDataReader](#) . [GetBytes](#) Method

Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetBytes( _  
    ByVal i As Integer, _  
    ByVal dataIndex As Long, _  
    ByVal buffer As Byte(), _  
    ByVal bufferIndex As Integer, _  
    ByVal length As Integer _  
    ) As Long _  
    Implements IDataRecord.GetBytes
```

Syntax: C#

```
public long GetBytes(  
    inti,  
    longdataIndex,  
    byte[]buffer,  
    intbufferIndex,  
    intlength  
);
```

Parameters

- **i**: The zero-based column ordinal.
- **dataIndex**: The index within the field from which to begin the read operation.
- **buffer**: The buffer into which to read the stream of bytes.
- **bufferIndex**: The index for buffer to begin the read operation.
- **length**: The maximum length to copy into the buffer.

Return Value

The actual number of bytes read.

Implements

IDataRecord.GetBytes

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.11. [MySqlDataReader.GetChar](#) Method

Gets the value of the specified column as a single character.

Syntax: Visual Basic

```
NotOverridable Public Function GetChar( _  
    ByVal i As Integer _  
) As Char _  
    Implements IDataRecord.GetChar
```

Syntax: C#

```
public char GetChar(  
    inti  
);
```

Parameters

- **i**:

Return Value**Implements**

IDataRecord.GetChar

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.12. [MySqlDataReader.GetChars](#) Method

Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetChars( _  
    ByVal i As Integer, _  
    ByVal fieldOffset As Long, _  
    ByVal buffer As Char(), _  
    ByVal bufferoffset As Integer, _  
    ByVal length As Integer _  
) As Long _  
    Implements IDataRecord.GetChars
```

Syntax: C#

```
public long GetChars(  
    inti,  
    longfieldOffset,  
    char[]buffer,  
    intbufferoffset,  
    intlength  
);
```


Parameters

- `i`:
- `fieldOffset`:
- `buffer`:
- `bufferoffset`:
- `length`:

Return Value**Implements**

IDataRecord.GetChars

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.13. [MySQLDataReader.GetDataTypeName](#) Method

Gets the name of the source data type.

Syntax: Visual Basic

```
NotOverridable Public Function GetDataTypeName( _  
    ByVal i As Integer _  
) As String _  
- Implements IDataRecord.GetDataTypeName
```

Syntax: C#

```
public string GetDataTypeName(  
    inti  
);
```

Parameters

- `i`:

Return Value**Implements**

IDataRecord.GetDataTypeName

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.14. [MySQLDataReader.GetDateTime](#) Method

Syntax: Visual Basic

```
NotOverridable Public Function GetDateTime( _  
    ByVal index As Integer _  
) As Date _  
- Implements IDataRecord.GetDateTime
```

Syntax: C#

```
public DateTime GetDateTime(  
    int index  
);
```

Implements

IDataRecord.GetDateTime

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.15. [MySQLDataReader.GetDecimal](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetDecimal( _  
    ByVal index As Integer _  
) As Decimal _  
    Implements IDataRecord.GetDecimal
```

Syntax: C#

```
public decimal GetDecimal(  
    int index  
);
```

Implements

IDataRecord.GetDecimal

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.16. [MySQLDataReader.GetDouble](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetDouble( _  
    ByVal index As Integer _  
) As Double _  
    Implements IDataRecord.GetDouble
```

Syntax: C#

```
public double GetDouble(  
    int index  
);
```

Implements

IDataRecord.GetDouble

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.17. [MySQLDataReader.GetFieldType](#) Method

Gets the Type that is the data type of the object.

Syntax: Visual Basic

```
NotOverridable Public Function GetFieldType( _  
    ByVal i As Integer _  
) As Type _  
- Implements IDataRecord.GetFieldType
```

Syntax: C#

```
public Type GetFieldType(  
    inti  
);
```

Parameters

- **i:**

Return Value**Implements**

IDataRecord.GetFieldType

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.18. [MySQLDataReader.GetFloat](#) Method

Syntax: Visual Basic

```
NotOverridable Public Function GetFloat( _  
    ByVal index As Integer _  
) As Single _  
- Implements IDataRecord.GetFloat
```

Syntax: C#

```
public float GetFloat(  
    intindex  
);
```

Implements

IDataRecord.GetFloat

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.19. [MySQLDataReader.GetGuid](#) Method

Syntax: Visual Basic

```
NotOverridable Public Function GetGuid( _  
    ByVal index As Integer _  
) As Guid _  
- Implements IDataRecord.GetGuid
```

Syntax: C#

```
public Guid GetGuid(  
    intindex  
);
```

Implements

IDataRecord.GetGuid

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.20. [MySqlDataReader.GetInt16](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetInt16( _  
    ByVal index As Integer _  
) As Short _  
    Implements IDataRecord.GetInt16
```

Syntax: C#

```
public short GetInt16(  
    int index  
);
```

Implements

IDataRecord.GetInt16

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.21. [MySqlDataReader.GetInt32](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetInt32( _  
    ByVal index As Integer _  
) As Integer _  
    Implements IDataRecord.GetInt32
```

Syntax: C#

```
public int GetInt32(  
    int index  
);
```

Implements

IDataRecord.GetInt32

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.22. [MySqlDataReader.GetInt64](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetInt64( _  
    ByVal index As Integer _  
) As Long _  
    Implements IDataRecord.GetInt64
```

Syntax: C#

```
public long GetInt64(  

```

```
intindex  
);
```

Implements

IDataRecord.GetInt64

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.23. [MySQLDataReader.GetMySqlDateTime](#) Method

Syntax: Visual Basic

```
Public Function GetMySqlDateTime( _  
    ByVal index As Integer _  
) As MySqlDateTime
```

Syntax: C#

```
public MySqlDateTime GetMySqlDateTime(  
intindex  
);
```

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.24. [MySQLDataReader.GetName](#) Method

Gets the name of the specified column.

Syntax: Visual Basic

```
NotOverridable Public Function GetName( _  
    ByVal i As Integer _  
) As String _  
    Implements IDataRecord.GetName
```

Syntax: C#

```
public string GetName(  
inti  
);
```

Parameters

- [i](#):

Return Value**Implements**

IDataRecord.GetName

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.25. [MySQLDataReader.GetOrdinal](#) Method

Gets the column ordinal, given the name of the column.

Syntax: Visual Basic

```
NotOverridable Public Function GetOrdinal( _  
    ByVal name As String _  
    ) As Integer _  
    Implements IDataRecord.GetOrdinal
```

Syntax: C#

```
public int GetOrdinal(  
    stringname  
);
```

Parameters

- **name:**

Return Value**Implements**

IDataRecord.GetOrdinal

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.26. [MySQLDataReader.GetSchemaTable](#) Method

Returns a DataTable that describes the column metadata of the MySQLDataReader.

Syntax: Visual Basic

```
NotOverridable Public Function GetSchemaTable() As DataTable _  
    Implements IDataReader.GetSchemaTable
```

Syntax: C#

```
public DataTable GetSchemaTable();
```

Return Value**Implements**

IDataReader.GetSchemaTable

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.27. [MySQLDataReader.GetString](#) Method**Syntax: Visual Basic**

```
NotOverridable Public Function GetString( _  
    ByVal index As Integer _  
    ) As String _  
    Implements IDataRecord.GetString
```

Syntax: C#

```
public string GetString(  
    intindex
```

```
);
```

Implements

IDataRecord.GetString

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.28. [MySqlDataReader.GetString](#) Method

Syntax: Visual Basic

```
Public Function GetString( _  
    ByVal index As Integer _  
) As String
```

Syntax: C#

```
public String GetString(  
    int index  
);
```

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.29. [MySqlDataReader.GetInt16](#) Method

Syntax: Visual Basic

```
Public Function GetInt16( _  
    ByVal index As Integer _  
) As Int16
```

Syntax: C#

```
public short GetInt16(  
    int index  
);
```

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.30. [MySqlDataReader.GetInt32](#) Method

Syntax: Visual Basic

```
Public Function GetInt32( _  
    ByVal index As Integer _  
) As Int32
```

Syntax: C#

```
public int GetInt32(  
    int index  
);
```

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.31. [MySqlDataReader.GetInt64](#) Method

Syntax: Visual Basic

```
Public Function GetUInt64( _  
    ByVal index As Integer _  
) As UInt64
```

Syntax: C#

```
public ulong GetUInt64(  
    int index  
);
```

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.32. [MySQLDataReader.GetValue](#) Method

Gets the value of the specified column in its native format.

Syntax: Visual Basic

```
NotOverridable Public Function GetValue( _  
    ByVal i As Integer _  
) As Object _  
    Implements IDataRecord.GetValue
```

Syntax: C#

```
public object GetValue(  
    int i  
);
```

Parameters

- **i**:

Return Value**Implements**

IDataRecord.GetValue

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.33. [MySQLDataReader.GetValues](#) Method

Gets all attribute columns in the collection for the current row.

Syntax: Visual Basic

```
NotOverridable Public Function GetValues( _  
    ByVal values As Object() _  
) As Integer _  
    Implements IDataRecord.GetValues
```

Syntax: C#

```
public int GetValues(  
    object[] values  
);
```


Parameters

- `values:`

Return Value**Implements**

`IDataRecord.GetValues`

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.34. [MySqlDataReader.IsDBNull](#) Method

Gets a value indicating whether the column contains non-existent or missing values.

Syntax: Visual Basic

```
NotOverridable Public Function IsDBNull( _  
    ByVal i As Integer _  
) As Boolean _  
    Implements IDataRecord.IsDBNull
```

Syntax: C#

```
public bool IsDBNull(  
    int i  
);
```

Parameters

- `i:`

Return Value**Implements**

`IDataRecord.IsDBNull`

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.1.35. [MySqlDataReader.NextResult](#) Method

Advances the data reader to the next result, when reading the results of batch SQL statements.

Syntax: Visual Basic

```
NotOverridable Public Function NextResult() As Boolean _  
    Implements IDataReader.NextResult
```

Syntax: C#

```
public bool NextResult();
```

Return Value**Implements**

IDataReader.NextResult

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.1.1.36. [MySQLDataReader.Read](#) Method

Advances the MySQLDataReader to the next record.

Syntax: Visual Basic

```
NotOverridable Public Function Read() As Boolean _  
    Implements IDataReader.Read
```

Syntax: C#

```
public bool Read();
```

Return Value

Implements

IDataReader.Read

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.13.2. [MySQLCommand.ExecuteReader](#) Method

Syntax: Visual Basic

```
Overloads Public Function ExecuteReader( _  
    ByVal behavior As CommandBehavior _  
) As MySQLDataReader
```

Syntax: C#

```
public MySQLDataReader ExecuteReader(  
    CommandBehavior behavior  
);
```

See Also

[MySQLCommand Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLCommand.ExecuteReader Overload List](#)

26.2.4.1.2.1.14. [MySQLCommand.ExecuteScalar](#) Method

Syntax: Visual Basic

```
NotOverridable Public Function ExecuteScalar() As Object _  
    Implements IDbCommand.ExecuteScalar
```

Syntax: C#

```
public object ExecuteScalar();
```

Implements

IDbCommand.ExecuteScalar

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.2.1.15. MySqlCommand.Prepare Method

Syntax: Visual Basic

```
NotOverridable Public Sub Prepare() _  
    Implements IDbCommand.Prepare
```

Syntax: C#

```
public void Prepare();
```

Implements

IDbCommand.Prepare

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3. MySqlCommandBuilder Class

For a list of all members of this type, see [MySqlCommandBuilder Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommandBuilder_  
    Inherits Component
```

Syntax: C#

```
public sealed class MySqlCommandBuilder : Component
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlCommandBuilder Members](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1. MySqlCommandBuilder Members

[MySqlCommandBuilder overview](#)

Public Static (Shared) Methods

 DeriveParameters 	Overloaded. Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

Public Instance Constructors

MySQLCommandBuilder	Overloaded. Initializes a new instance of the MySqlCommand-Builder class.

Public Instance Properties

Container(inherited from Component)	Gets the IContainerthat contains the Component.
DataAdapter	
QuotePrefix	
QuoteSuffix	
Site(inherited from Component)	Gets or sets the ISiteof the Component.

Public Instance Methods

CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information re-quired to generate a proxy used to communicate with a remote ob-ject.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Ob-ject.
GetDeleteCommand	
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetInsertCommand	
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the life-time policy for this instance.
GetType(inherited from Object)	Gets the Typeof the current instance.
GetUpdateCommand	
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
RefreshSchema	
ToString(inherited from Component)	Returns a Stringcontaining the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposedevent on the com-ponent.

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.1. DeriveParameters Method

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.

Overload List

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.

- [public static void DeriveParameters\(MySqlCommand\);](#)
- [public static void DeriveParameters\(MySqlCommand,bool\);](#)

See Also

[MySqlCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.1.1. MySqlCommandBuilder.DeriveParameters Method

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _  
    ByVal command As MySqlCommand _  
)
```

Syntax: C#

```
public static void DeriveParameters(  
    MySqlCommandcommand  
);
```

Parameters

- **command**: The MySqlCommand referencing the stored procedure from which the parameter information is to be derived. The derived parameters are added to the Parameters collection of the MySqlCommand.

Exceptions

Exception Type	Condition
InvalidOperationException	The command text is not a valid stored procedure name.

See Also

[MySqlCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySqlCommandBuilder.DeriveParameters Overload List](#)

26.2.4.1.3.1.1.2. MySqlCommandBuilder.DeriveParameters Method

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _  
    ByVal command As MySqlCommand, _  
    ByVal useProc As Boolean _  
)
```

Syntax: C#

```
public static void DeriveParameters(  
    MySqlCommandcommand,  
    booluseProc  
);
```

See Also

[MySqlCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySqlCommandBuilder.DeriveParameters Overload List](#)

26.2.4.1.3.1.2. MySqlCommandBuilder Constructor

Initializes a new instance of the [MySqlCommandBuilder](#) class.

Overload List

Initializes a new instance of the [MySQLCommandBuilder](#) class.

- [public MySQLCommandBuilder\(\);](#)
- [public MySQLCommandBuilder\(MySqlDataAdapter\);](#)
- [public MySQLCommandBuilder\(MySqlDataAdapter,bool\);](#)
- [public MySQLCommandBuilder\(bool\);](#)

See Also

[MySQLCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.1. [MySQLCommandBuilder](#) Constructor

Initializes a new instance of the [MySQLCommandBuilder](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySQLCommandBuilder();
```

See Also

[MySQLCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySQLCommandBuilder Constructor Overload List](#)

26.2.4.1.3.1.2.2. [MySQLCommandBuilder](#) Constructor**Syntax: Visual Basic**

```
Overloads Public Sub New( _  
    ByVal adapter As MySqlDataAdapter _  
)
```

Syntax: C#

```
public MySQLCommandBuilder(  
    MySqlDataAdapteradapter  
);
```

See Also

[MySQLCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySQLCommandBuilder Constructor Overload List](#)

26.2.4.1.3.1.2.2.1. [MySqlDataAdapter](#) Class

For a list of all members of this type, see [MySqlDataAdapter Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataAdapter_  
    Inherits DbDataAdapter
```

Syntax: C#

```
public sealed class MySqlDataAdapter : DbDataAdapter
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlDataAdapter Members](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1. [MySqlDataAdapter](#) Members

[MySqlDataAdapter overview](#)

Public Instance Constructors

MySqlDataAdapter	Overloaded. Initializes a new instance of the MySqlDataAdapter class.

Public Instance Properties

AcceptChangesDuringFill (inherited from DataAdapter)	Gets or sets a value indicating whether AcceptChanges is called on a DataRow after it is added to the DataTable during any of the Fill operations.
AcceptChangesDuringUpdate (inherited from DataAdapter)	Gets or sets whether AcceptChanges is called during a Update .
Container (inherited from Component)	Gets the IContainer that contains the Component .
ContinueUpdateOnError (inherited from DataAdapter)	Gets or sets a value that specifies whether to generate an exception when an error is encountered during a row update.
DeleteCommand	Overloaded.
FillLoadOption (inherited from DataAdapter)	Gets or sets the LoadOption that determines how the adapter fills the DataTable from the DbDataReader .
InsertCommand	Overloaded.
MissingMappingAction (inherited from DataAdapter)	Determines the action to take when incoming data does not have a matching table or column.
MissingSchemaAction (inherited from DataAdapter)	Determines the action to take when existing DataSet schema does not match incoming data.
ReturnProviderSpecificTypes (inherited from DataAdapter)	Gets or sets whether the Fill method should return provider-specific values or common CLS-compliant values.
SelectCommand	Overloaded.
Site (inherited from Component)	Gets or sets the ISite of the Component .
TableMappings (inherited from DataAdapter)	Gets a collection that provides the master mapping between a source table and a DataTable .
UpdateBatchSize (inherited from DbDataAdapter)	Gets or sets a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.
UpdateCommand	Overloaded.

Public Instance Methods

CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote ob-

	ject.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
Fill(inherited from DbDataAdapter)	Overloaded. Adds or refreshes rows in the DataSet to match those in the data source using the DataSetname, and creates a DataTable named "Table."
FillSchema(inherited from DbDataAdapter)	Overloaded. Configures the schema of the specified DataTable based on the specified SchemaType.
GetFillParameters(inherited from DbDataAdapter)	Gets the parameters set by the user when executing an SQL SELECT statement.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ResetFillLoadOption(inherited from DataAdapter)	Resets FillLoadOption to its default state and causes Fill to honor AcceptChangesDuringFill.
ShouldSerializeAcceptChangesDuringFill(inherited from DataAdapter)	Determines whether the AcceptChangesDuringFill property should be persisted.
ShouldSerializeFillLoadOption(inherited from DataAdapter)	Determines whether the FillLoadOption property should be persisted.
ToString(inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.
Update(inherited from DbDataAdapter)	Overloaded. Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
FillError(inherited from DataAdapter)	Returned when an error occurs during a fill operation.
RowUpdated	Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.
RowUpdating	Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Protected Internal Instance Properties

FillCommandBehavior(inherited from DbDataAdapter)	Gets or sets the behavior of the command used to fill the data adapter.

See Also

[MySQLDataAdapter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.1. [MySQLDataAdapter](#) Constructor

Initializes a new instance of the [MySQLDataAdapter](#) class.

Overload List

Initializes a new instance of the [MySqlDataAdapter](#) class.

- `public MySQLDataAdapter();`
- `public MySQLDataAdapter(MySqlCommand);`
- `public MySQLDataAdapter(string, MySqlConnection);`
- `public MySQLDataAdapter(string, string);`

See Also

MySqlDataAdapter Class , MySql.Data.MySqlClient Namespace

26.2.4.1.3.1.2.2.1.1.1.1.1. MySQLDataAdapter Constructor

Initializes a new instance of the [MySqlDataAdapter](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlDataAdapter();
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlDataAdapter Constructor Overload List](#)

26.2.4.1.3.1.2.2.1.1.1.2. `MySqlDataAdapter` Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal selectCommand As MySqlCommand _  
)
```

Syntax: C#

```
public MySqlDataAdapter(
    MySqlCommand selectCommand
);
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlDataAdapter Constructor Overload List](#)

26.2.4.1.3.1.2.2.1.1.1.3. MySqlConnection Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal selectCommandText As String, _
    ByVal connection As MySqlConnection _
)
```

Syntax: C#

```
public MySqlDataAdapter(
stringselectCommandText,
MySqlConnectionconnection
);
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlDataAdapter Constructor Overload List](#)

26.2.4.1.3.1.2.2.1.1.4. [MySqlDataAdapter](#) Constructor**Syntax: Visual Basic**

```
Overloads Public Sub New( _  
    ByVal selectCommandText As String, _  
    ByVal selectConnString As String _  
)
```

Syntax: C#

```
public MySqlDataAdapter(  
    stringselectCommandText,  
    stringselectConnString  
);
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlDataAdapter Constructor Overload List](#)

26.2.4.1.3.1.2.2.1.1.2. DeleteCommand Property

Syntax: Visual Basic

```
Overloads Public Property DeleteCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand DeleteCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.3. InsertCommand Property

Syntax: Visual Basic

```
Overloads Public Property InsertCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand InsertCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.4. SelectCommand Property

Syntax: Visual Basic

```
Overloads Public Property SelectCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand SelectCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.5. UpdateCommand Property

Syntax: Visual Basic

```
Overloads Public Property UpdateCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand UpdateCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.6. [MySqlDataAdapter](#) .RowUpdated Event

Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdated As MySqlRowUpdatedEventHandler
```

Syntax: C#

```
public event MySqlRowUpdatedEventHandler RowUpdated;
```

Event Data

The event handler receives an argument of type [MySqlRowUpdatedEventArgs](#) containing data related to this event. The following [MySqlRowUpdatedEventArgs](#) properties provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand executed when Update is called.
Errors	Gets any errors generated by the .NET Framework data provider when the Command was executed.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row	Gets the DataRow sent through an Update.
RowCount	Gets the number of rows processed in a batch of updated records.
StatementType	Gets the type of SQL statement executed.
Status	Gets the UpdateStatus of the Command property.
TableMapping	Gets the DataTableMapping sent through an Update.

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.6.1. [MySqlRowUpdatedEventHandler](#) Delegate

Represents the method that will handle the RowUpdated event of a [MySqlDataAdapter](#) .

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatedEventHandler( _  
    ByVal sender As Object, _  
    ByVal e As MySqlRowUpdatedEventArgs _  
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatedEventHandler(
    object sender,
    MySqlRowUpdatedEventArgs
);
```

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.6.1.1. [MySqlRowUpdatedEventArgs](#) Class

Provides data for the RowUpdated event. This class cannot be inherited.

For a list of all members of this type, see [MySqlRowUpdatedEventArgs Members](#).

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatedEventArgs_
    Inherits RowUpdatedEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatedEventArgs : RowUpdatedEventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlRowUpdatedEventArgs Members](#), [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.6.1.1.1. [MySqlRowUpdatedEventArgs](#) Members

[MySqlRowUpdatedEventArgs overview](#)

Public Instance Constructors

MySqlRowUpdatedEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatedEventArgs class.

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand executed when Update is called.
Errors(inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the Command was executed.
RecordsAffected(inherited from RowUpdatedEventArgs)	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Row(inherited from RowUpdatedEventArgs)	Gets the DataRow sent through an Update.
RowCount(inherited from RowUpdatedEventArgs)	Gets the number of rows processed in a batch of updated records.
StatementType(inherited from RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status(inherited from RowUpdatedEventArgs)	Gets the UpdateStatus of the Command property.
TableMapping(inherited from RowUpdatedEventArgs)	Gets the DataTableMapping sent through an Update.

Public Instance Methods

CopyToRows(inherited from RowUpdatedEventArgs)	Overloaded. Copies references to the modified rows into the provided array.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Type of the current instance.
ToString(inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLRowUpdatedEventArgs Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.6.1.1.1.1. MySQLRowUpdatedEventArgs Constructor

Initializes a new instance of the MySQLRowUpdatedEventArgs class.

Syntax: Visual Basic

```
Public Sub New( _
    ByVal row As DataRow, _
    ByVal command As IDbCommand, _
    ByVal statementType As StatementType, _
    ByVal tableMapping As DataTableMapping _
)
```

Syntax: C#

```
public MySQLRowUpdatedEventArgs(
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
);
```

Parameters

- **row**: The DataRow sent through an Update.
- **command**: The IDbCommand executed when Update is called.
- **statementType**: One of the StatementType values that specifies the type of query executed.
- **tableMapping**: The DataTableMapping sent through an Update.

See Also

[MySQLRowUpdatedEventArgs Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.6.1.1.1.2. Command Property

Gets or sets the MySqlCommand executed when Update is called.

Syntax: Visual Basic

```
Overloads Public ReadOnly Property Command As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand Command {get;}
```

See Also

[MySqlRowUpdatedEventArgs Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.7. [MySqlDataAdapter.RowUpdating](#) Event

Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdating As MySqlRowUpdatingEventHandler
```

Syntax: C#

```
public event MySqlRowUpdatingEventHandler RowUpdating;
```

Event Data

The event handler receives an argument of type [MySqlRowUpdatingEventArgs](#) containing data related to this event. The following [MySqlRowUpdatingEventArgs](#) properties provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand to execute when performing the Update.
Errors	Gets any errors generated by the .NET Framework data provider when the Command executes.
Row	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType	Gets the type of SQL statement to execute.
Status	Gets or sets the UpdateStatus of the Command property.
TableMapping	Gets the DataTableMapping to send through the Update.

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.7.1. [MySqlRowUpdatingEventHandler](#) Delegate

Represents the method that will handle the RowUpdating event of a [MySqlDataAdapter](#).

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatingEventHandler( _  
    ByVal sender As Object, _  
    ByVal e As MySqlRowUpdatingEventArgs _  
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatingEventHandler(  
    object sender,
```

```
MySQLRowUpdatingEventArgs
};
```

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.7.1.1.1. MySQLRowUpdatingEventArgs Class

Provides data for the RowUpdating event. This class cannot be inherited.

For a list of all members of this type, see [MySQLRowUpdatingEventArgs Members](#).

Syntax: Visual Basic

```
NotInheritable Public Class MySQLRowUpdatingEventArgs_
    Inherits RowUpdatingEventArgs
```

Syntax: C#

```
public sealed class MySQLRowUpdatingEventArgs : RowUpdatingEventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLRowUpdatingEventArgs Members](#), [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.7.1.1.1.1. MySQLRowUpdatingEventArgs Members

[MySQLRowUpdatingEventArgs overview](#)

Public Instance Constructors

MySQLRowUpdatingEventArgs Constructor	Initializes a new instance of the MySQLRowUpdatingEventArgs class.

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand to execute when performing the Update.
Errors(inherited from RowUpdatingEventArgs)	Gets any errors generated by the .NET Framework data provider when the Command executes.
Row(inherited from RowUpdatingEventArgs)	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType(inherited from RowUpdatingEventArgs)	Gets the type of SQL statement to execute.
Status(inherited from RowUpdatingEventArgs)	Gets or sets the UpdateStatus of the Command property.

TableMapping(inherited from RowUpdatingEventArgs)	Gets the DataTableMapping to send through the Update.

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Type of the current instance.
ToString(inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLRowUpdatingEventArgs Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.7.1.1.1.1. MySQLRowUpdatingEventArgs Constructor

Initializes a new instance of the MySQLRowUpdatingEventArgs class.

Syntax: Visual Basic

```
Public Sub New( _
    ByVal row As DataRow, _
    ByVal command As IDbCommand, _
    ByVal statementType As StatementType, _
    ByVal tableMapping As DataTableMapping _
)
```

Syntax: C#

```
public MySQLRowUpdatingEventArgs(
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
);
```

Parameters

- **row**: The DataRow to update.
- **command**: The IDbCommand to execute during update.
- **statementType**: One of the StatementType values that specifies the type of query executed.
- **tableMapping**: The DataTableMapping sent through an update.

See Also

[MySQLRowUpdatingEventArgs Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.2.1.1.7.1.1.1.2. Command Property

Gets or sets the MySqlCommand to execute when performing the update.

Syntax: Visual Basic

```
Overloads Public Property Command As MySqlCommand
```

Syntax: C#


```
new public MySqlCommand Command {get; set;}
```

See Also

[MySQLRowUpdatingEventArgs Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.2.3. [MySQLCommandBuilder](#) Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal adapter As MySqlDataAdapter, _  
    ByVal lastOneWins As Boolean _  
)
```

Syntax: C#

```
public MySqlCommandBuilder(  
    MySqlDataAdapteradapter,  
    boollastOneWins  
);
```

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLCommandBuilder Constructor Overload List](#)

26.2.4.1.3.1.2.4. [MySQLCommandBuilder](#) Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal lastOneWins As Boolean _  
)
```

Syntax: C#

```
public MySqlCommandBuilder(  
    boollastOneWins  
);
```

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLCommandBuilder Constructor Overload List](#)

26.2.4.1.3.1.3. DataAdapter Property

Syntax: Visual Basic

```
Public Property DataAdapter As MySqlDataAdapter
```

Syntax: C#

```
public MySqlDataAdapter DataAdapter {get; set;}
```

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.4. QuotePrefix Property

Syntax: Visual Basic

```
Public Property QuotePrefix As String
```

Syntax: C#

```
public string QuotePrefix {get; set;}
```

See Also

[MySqlCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.5. QuoteSuffix Property

Syntax: Visual Basic

```
Public Property QuoteSuffix As String
```

Syntax: C#

```
public string QuoteSuffix {get; set;}
```

See Also

[MySqlCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.6. MySqlCommandBuilder.GetDeleteCommand Method

Syntax: Visual Basic

```
Public Function GetDeleteCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetDeleteCommand();
```

See Also

[MySqlCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.7. MySqlCommandBuilder.GetInsertCommand Method

Syntax: Visual Basic

```
Public Function GetInsertCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetInsertCommand();
```

See Also

[MySqlCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.8. MySqlCommandBuilder.GetUpdateCommand Method

Syntax: Visual Basic

```
Public Function GetUpdateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetUpdateCommand();
```

See Also

[MySqlCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.3.1.9. MySqlCommandBuilder.RefreshSchema Method

Syntax: Visual Basic

```
Public Sub RefreshSchema()
```

Syntax: C#

```
public void RefreshSchema();
```

See Also

[MySqlCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.4. MySqlConnectionException Class

The exception that is thrown when MySQL returns an error. This class cannot be inherited.

For a list of all members of this type, see [MySqlConnectionException Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnectionException_  
    Inherits SystemException
```

Syntax: C#

```
public sealed class MySqlConnectionException : SystemException
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlConnection](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySqlConnectionException Members](#) , [MySQL.Data.MySqlConnection Namespace](#)

26.2.4.1.4.1. MySqlConnectionException Members

[MySqlConnectionException overview](#)

Public Instance Properties

Data(inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink(inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException(inherited from Exception)	Gets the Exception instance that caused the current exception.
Message(inherited from Exception)	Gets a message that describes the current exception.
Number	Gets a number that identifies the type of error.
Source(inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace(inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite(inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException(inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData(inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType(inherited from Exception)	Gets the runtime type of the current instance.
ToString(inherited from Exception)	Creates and returns a string representation of the current exception.

See Also

[MySqlException Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.4.1.1. Number Property

Gets a number that identifies the type of error.

Syntax: Visual Basic

```
Public ReadOnly Property Number As Integer
```

Syntax: C#

```
public int Number {get;}
```

See Also

[MySqlException Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.5. MySqlHelper Class

Helper class that makes it easier to work with the provider.

For a list of all members of this type, see [MySqlHelper Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlHelper
```

Syntax: C#

```
public sealed class MySqlHelper
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySQLHelper Members](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.5.1. MySQLHelper Members

[MySQLHelper overview](#)

Public Static (Shared) Methods

ExecuteDataRow	Executes a single SQL command and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteDataset	Overloaded. Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteNonQuery	Overloaded. Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.
ExecuteReader	Overloaded. Executes a single command against a MySQL database.
ExecuteScalar	Overloaded. Execute a single command against a MySQL database.
UpdateDataSet	Updates the given table with data from the given DataSet

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.5.1.1. MySQLHelper.ExecuteDataRow Method

Executes a single SQL command and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Public Shared Function ExecuteDataRow( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ParamArray parms As MySqlParameter() _
) As DataRow
```

Syntax: C#

```
public static DataRow ExecuteDataRow(
    string connectionString,
    string commandText,
    params MySqlParameter[] parms
);
```

Parameters

- **connectionString**: Settings to be used for the connection
- **commandText**: Command to execute
- **parms**: Parameters to use for the command

Return Value

DataRow containing the first row of the resultset

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.5.1.2. ExecuteDataset Method

Executes a single SQL command and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

Overload List

Executes a single SQL command and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

- `public static DataSet ExecuteDataset(MySqlConnection,string);`

Executes a single SQL command and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

- `public static DataSet ExecuteDataset(MySqlConnection,string,params MySqlParameter[]);`

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- `public static DataSet ExecuteDataset(string,string);`

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- `public static DataSet ExecuteDataset(string,string,params MySqlParameter[]);`

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.5.1.2.1. `MySqlHelper.ExecuteDataset` Method

Executes a single SQL command and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
    MySqlConnection connection,  
    string commandText  
);
```

Parameters

- **connection**: [MySqlConnection](#) object to use
- **commandText**: Command to execute

Return Value

DataSet containing the resultset

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLHelper.ExecuteDataset Overload List](#)

26.2.4.1.5.1.2.2. [MySQLHelper.ExecuteDataset](#) Method

Executes a single SQL command and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
    MySqlConnection connection,  
    string commandText,  
    params MySqlParameter[] commandParameters  
);
```

Parameters

- **connection**: [MySqlConnection](#) object to use
- **commandText**: Command to execute
- **commandParameters**: Parameters to use for the command

Return Value

DataSet containing the resultset

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLHelper.ExecuteDataset Overload List](#)

26.2.4.1.5.1.2.3. [MySQLHelper.ExecuteDataset](#) Method

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connectionString As String, _
```

```
ByVal commandText As String _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
    string connectionString,  
    string commandText  
);
```

Parameters

- **connectionString**: Settings to be used for the connection
- **commandText**: Command to execute

Return Value

DataSet containing the resultset

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLHelper.ExecuteDataset Overload List](#)

26.2.4.1.5.1.2.4. [MySQLHelper.ExecuteDataset](#) Method

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
    string connectionString,  
    string commandText,  
    params MySqlParameter[] commandParameters  
);
```

Parameters

- **connectionString**: Settings to be used for the connection
- **commandText**: Command to execute
- **commandParameters**: Parameters to use for the command

Return Value

DataSet containing the resultset

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLHelper.ExecuteDataset Overload List](#)

26.2.4.1.5.1.3. [ExecuteNonQuery](#) Method

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and re-

mains open after the method completes.

Overload List

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

- [public static int ExecuteNonQuery\(MySqlConnection,string,params MySqlParameter\[\]\);](#)

Executes a single command against a MySQL database. A new [MySqlConnection](#) is created using the [ConnectionString](#) given.

- [public static int ExecuteNonQuery\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.5.1.3.1. [MySqlHelper.ExecuteNonQuery](#) Method

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
    ) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(  
    MySqlConnectionconnection,  
    stringcommandText,  
    params MySqlParameter[]commandParameters  
);
```

Parameters

- [connection](#): [MySqlConnection](#) object to use
- [commandText](#): SQL command to be executed
- [commandParameters](#): Array of [MySqlParameter](#) objects to use with the command.

Return Value

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlHelper.ExecuteNonQuery Overload List](#)

26.2.4.1.5.1.3.2. [MySqlHelper.ExecuteNonQuery](#) Method

Executes a single command against a MySQL database. A new [MySqlConnection](#) is created using the [ConnectionString](#) given.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray parms As MySqlParameter() _  
    ) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(
    string connectionString,
    string commandText,
    params MySqlParameter[] params
);
```

Parameters

- **connectionString**: [ConnectionString](#) to use
- **commandText**: SQL command to be executed
- **params**: Array of [MySqlParameter](#) objects to use with the command.

Return Value**See Also**

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlHelper.ExecuteNonQuery Overload List](#)

26.2.4.1.5.1.4. ExecuteReader Method

Executes a single command against a MySQL database.

Overload List

Executes a single command against a MySQL database.

- [public static MySqlDataReader ExecuteReader\(string,string\);](#)

Executes a single command against a MySQL database.

- [public static MySqlDataReader ExecuteReader\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.5.1.4.1. [MySqlHelper.ExecuteReader](#) Method

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _
    ByVal connectionString As String, _
    ByVal commandText As String _
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(
    string connectionString,
    string commandText
);
```

Parameters

- **connectionString**: Settings to use for this command

- [commandText](#): Command text to use

Return Value

[MySQLDataReader](#) object ready to read the results of the command

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLHelper.ExecuteReader Overload List](#)

26.2.4.1.5.1.4.2. [MySQLHelper.ExecuteReader](#) Method

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(  
    string connectionString,  
    string commandText,  
    params MySqlParameter[] commandParameters  
);
```

Parameters

- [connectionString](#): Settings to use for this command
- [commandText](#): Command text to use
- [commandParameters](#): Array of [MySqlParameter](#) objects to use with the command

Return Value

[MySQLDataReader](#) object ready to read the results of the command

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLHelper.ExecuteReader Overload List](#)

26.2.4.1.5.1.5. ExecuteScalar Method

Execute a single command against a MySQL database.

Overload List

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(MySqlConnection,string\);](#)

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(MySqlConnection,string,params MySqlParameter\[\]\);](#)

Execute a single command against a MySQL database.

- `public static object ExecuteScalar(string,string);`

Execute a single command against a MySQL database.

- `public static object ExecuteScalar(string,string,params MySqlParameter[]);`

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#)

26.2.4.1.5.1.5.1. [MySqlHelper.ExecuteScalar](#) Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
    MySqlConnectionconnection,  
    stringcommandText  
);
```

Parameters

- `connection`: [MySqlConnection](#) object to use
- `commandText`: Command text to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlHelper.ExecuteScalar Overload List](#)

26.2.4.1.5.1.5.2. [MySqlHelper.ExecuteScalar](#) Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
    MySqlConnectionconnection,  
    stringcommandText,  
    params MySqlParameter[]commandParameters  
);
```

Parameters

- **connection**: [MySQLConnection](#) object to use
- **commandText**: Command text to use for the command
- **commandParameters**: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLHelper.ExecuteScalar Overload List](#)

26.2.4.1.5.1.5.3. [MySQLHelper.ExecuteScalar](#) Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connectionString As String, _  
    ByVal commandText As String _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
string connectionString,  
string commandText  
);
```

Parameters

- **connectionString**: Settings to use for the update
- **commandText**: Command text to use for the update

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLHelper.ExecuteScalar Overload List](#)

26.2.4.1.5.1.5.4. [MySQLHelper.ExecuteScalar](#) Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
string connectionString,  
string commandText,  
params MySqlParameter[] commandParameters  
);
```

Parameters

- **connectionString**: Settings to use for the command
- **commandText**: Command text to use for the command
- **commandParameters**: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLHelper.ExecuteScalar Overload List](#)

26.2.4.1.5.1.6. MySQLHelper.UpdateDataSet Method

Updates the given table with data from the given DataSet

Syntax: Visual Basic

```
Public Shared Sub UpdateDataSet( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ByVal ds As DataSet, _  
    ByVal tablename As String _  
)
```

Syntax: C#

```
public static void UpdateDataSet(  
stringconnectionString,  
stringcommandText,  
DataSetds,  
stringtablename  
);
```

Parameters

- **connectionString**: Settings to use for the update
- **commandText**: Command text to use for the update
- **ds**: DataSetcontaining the new data to use in the update
- **tablename**: Tablename in the dataset to update

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#)

26.2.4.1.6. MySQLErrorCode Enumeration**Syntax: Visual Basic**

```
Public Enum MySQLErrorCode
```

Syntax: C#

```
public enum MySQLErrorCode
```

Members

Member Name	Description
PacketTooLarge	
PasswordNotAllowed	
DuplicateKeyEntry	
HostNotPrivileged	
PasswordNoMatch	
AnonymousUser	
DuplicateKey	
KeyNotFound	
DuplicateKeyName	

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

26.2.4.2. MySql.Data.Types

[Namespace hierarchy](#)

Classes

Class	Description
MySqlConversionException	Summary description for MySqlConversionException.
MySqlDateTime	Summary description for MySqlDateTime.
MySqlValue	

26.2.4.2.1. MySql.Data.TypesHierarchy**See Also**

[MySql.Data.Types Namespace](#)

26.2.4.2.2. MySqlConversionException Class

Summary description for MySqlConversionException.

For a list of all members of this type, see [MySqlConversionException Members](#).

Syntax: Visual Basic

```
Public Class MySqlConversionException_
    Inherits ApplicationException
```

Syntax: C#

```
public class MySqlConversionException : ApplicationException
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to

be thread-safe.

Requirements

Namespace: [MySQL.Data.Types](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLConversionException Members](#) , [MySQL.Data.Types Namespace](#)

26.2.4.2.2.1. MySQLConversionException Members

[MySQLConversionException overview](#)

Public Instance Constructors

MySQLConversionException Constructor	Ctor

Public Instance Properties

Data(inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink(inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException(inherited from Exception)	Gets the Exceptioninstance that caused the current exception.
Message(inherited from Exception)	Gets a message that describes the current exception.
Source(inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace(inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite(inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetBaseException(inherited from Exception)	When overridden in a derived class, returns the Exceptionthat is the root cause of one or more subsequent exceptions.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData(inherited from Exception)	When overridden in a derived class, sets the SerializationInfowith information about the exception.
GetType(inherited from Exception)	Gets the runtime type of the current instance.
ToString(inherited from Exception)	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult(inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.

Protected Instance Methods

Finalize(inherited from Object)	Allows an Objectto attempt to free resources and perform other

	cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySQLConversionException Class](#) , [MySQL.Data.Types Namespace](#)

26.2.4.2.2.1.1. [MySQLConversionException](#) Constructor

Syntax: Visual Basic

```
Public Sub New(  
    ByVal msg As String _  
)
```

Syntax: C#

```
public MySQLConversionException(  
    string msg  
) ;
```

See Also

[MySQLConversionException Class](#) , [MySQL.Data.Types Namespace](#)

26.2.4.2.3. [MySQLDateTime](#) Class

Summary description for MySQLDateTime.

For a list of all members of this type, see [MySQLDateTime Members](#) .

Syntax: Visual Basic

```
Public Class MySQLDateTime_  
    Inherits MySQLValue_  
    Implements IConvertible, IComparable
```

Syntax: C#

```
public class MySQLDateTime : MySQLValue, IConvertible, IComparable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.Types](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLDateTime Members](#) , [MySQL.Data.Types Namespace](#)

26.2.4.2.3.1. [MySQLDateTime](#) Members

[MySQLDateTime overview](#)

Public Static (Shared) Type Conversions

Explicit MySQLDateTime to DateTime Conversion	

Public Instance Properties

Day	Returns the day portion of this datetime
Hour	Returns the hour portion of this datetime
IsNull (inherited from MySQLValue)	
IsValidDateTime	Indicates if this object contains a value that can be represented as a DateTime
Minute	Returns the minute portion of this datetime
Month	Returns the month portion of this datetime
Second	Returns the second portion of this datetime
ValueAsObject (inherited from MySQLValue)	Returns the value of this field as an object
Year	Returns the year portion of this datetime

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object .
GetDateTime	Returns this value as a DateTime
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Returns a MySQL specific string representation of this value

Protected Instance Fields

classType (inherited from MySQLValue)	The system type represented by this value
dbType (inherited from MySQLValue)	The generic dbtype of this value
isNull (inherited from MySQLValue)	Is this value null
mysqlDbType (inherited from MySQLValue)	The specific MySQL db type
mysqlTypeName (inherited from MySQLValue)	The MySQL specific typename of this value
objectValue (inherited from MySQLValue)	

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object .

See Also

[MySQLDateTime Class](#) , [MySQL.Data.Types Namespace](#)

26.2.4.2.3.1.1. [MySQLDateTime](#) Explicit [MySQLDateTime](#) to [DateTime](#) Conversion

Syntax: Visual Basic

```
MySQLDateTime.op_Explicit(val)
```

Syntax: C#

```
public static explicit operator DateTime(  
    MySqlDateTime val  
);
```

Parameters

- `val`:

Return Value**See Also**

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.2. Day Property

Returns the day portion of this datetime

Syntax: Visual Basic

```
Public Property Day As Integer
```

Syntax: C#

```
public int Day {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.3. Hour Property

Returns the hour portion of this datetime

Syntax: Visual Basic

```
Public Property Hour As Integer
```

Syntax: C#

```
public int Hour {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4. IsNull Property

Syntax: Visual Basic

```
Public Property IsNull As Boolean
```

Syntax: C#

```
public bool IsNull {get; set;}
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1. [MySqlValue](#) Class

For a list of all members of this type, see [MySqlValue Members](#) .

Syntax: Visual Basic

```
MustInherit Public Class MySqlValue
```

Syntax: C#

```
public abstract class MySqlValue
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.Types](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlValue Members](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1.1. [MySqlValue](#) Members

[MySqlValue overview](#)

Protected Static (Shared) Fields

numberFormat	

Public Instance Constructors

MySqlValue Constructor	Initializes a new instance of the MySqlValue class.

Public Instance Properties

IsNull	
ValueAsObject	Returns the value of this field as an object

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Returns a string representation of this value

Protected Instance Fields

classType	The system type represented by this value
dbType	The generic dbtype of this value
isNull	Is this value null

mysqlDbType	The specific MySQL db type
mysqlTypeName	The MySQL specific typename of this value
objectValue	

Protected Instance Methods

Finalize(inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1.1.1. [MySqlValue.numberFormat](#) Field**Syntax: Visual Basic**

```
Protected Shared numberFormat As NumberFormatInfo
```

Syntax: C#

```
protected static NumberFormatInfo numberFormat;
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1.1.2. [MySqlValue](#) Constructor

Initializes a new instance of the [MySqlValue](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlValue();
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1.1.3. [ValueAsObject](#) Property

Returns the value of this field as an object

Syntax: Visual Basic

```
Public ReadOnly Property ValueAsObject As Object
```

Syntax: C#

```
public object ValueAsObject {get;}
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1.1.4. [MySqlValue.ToString](#) Method

Returns a string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1.1.5. [MySqlValue.classType](#) Field

The system type represented by this value

Syntax: Visual Basic

```
Protected classType As Type
```

Syntax: C#

```
protected Type classType;
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1.1.6. [MySqlValue.dbType](#) Field

The generic dbtype of this value

Syntax: Visual Basic

```
Protected dbType As DbType
```

Syntax: C#

```
protected DbType dbType;
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1.1.7. [MySqlValue.mysqlDbType](#) Field

The specific MySQL db type

Syntax: Visual Basic

```
Protected mysqlDbType As MySqlDbType
```

Syntax: C#

```
protected MySqlDbType mysqlDbType;
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1.1.8. [MySqlValue.mysqlTypeName](#) Field

The MySQL specific typename of this value

Syntax: Visual Basic

```
Protected mysqlTypeName As String
```

Syntax: C#

```
protected string mysqlTypeName;
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.4.1.1.9. [MySqlValue.objectValue](#) Field

Syntax: Visual Basic

```
Protected objectValue As Object
```

Syntax: C#

```
protected object objectValue;
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.5. [IsValidDateTime](#) Property

Indicates if this object contains a value that can be represented as a DateTime

Syntax: Visual Basic

```
Public ReadOnly Property IsValidDateTime As Boolean
```

Syntax: C#

```
public bool IsValidDateTime {get;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.6. [Minute](#) Property

Returns the minute portion of this datetime

Syntax: Visual Basic

```
Public Property Minute As Integer
```

Syntax: C#

```
public int Minute {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.7. Month Property

Returns the month portion of this datetime

Syntax: Visual Basic

```
Public Property Month As Integer
```

Syntax: C#

```
public int Month {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.8. Second Property

Returns the second portion of this datetime

Syntax: Visual Basic

```
Public Property Second As Integer
```

Syntax: C#

```
public int Second {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.9. Year Property

Returns the year portion of this datetime

Syntax: Visual Basic

```
Public Property Year As Integer
```

Syntax: C#

```
public int Year {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.10. [MySqlDateTime.GetDateTime](#) Method

Returns this value as a DateTime

Syntax: Visual Basic

```
Public Function GetDateTime() As Date
```

Syntax: C#

```
public DateTime GetDateTime();
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

26.2.4.2.3.1.11. `MySqlDateTime.ToString` Method

Returns a MySQL specific string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

26.2.5. Connector/NET Notes and Tips

In this section we will cover some of the more common use cases for Connector/NET, including BLOB handling, date handling, and using Connector/NET with common tools such as Crystal Reports.

26.2.5.1. Connecting to MySQL Using Connector/NET

26.2.5.1.1. Introduction

All interaction between a .NET application and the MySQL server is routed through a [MySqlConnection](#) object. Before your application can interact with the server, a [MySqlConnection](#) object must be instantiated, configured, and opened.

Even when using the [MySqlHelper](#) class, a [MySqlConnection](#) object is created by the helper class.

In this section, we will describe how to connect to MySQL using the [MySqlConnection](#) object.

26.2.5.1.2. Creating a Connection String

The [MySqlConnection](#) object is configured using a connection string. A connection string contains sever key/value pairs, separated by semicolons. Each key/value pair is joined with an equals sign.

The following is a sample connection string:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

In this example, the [MySqlConnection](#) object is configured to connect to a MySQL server at [127.0.0.1](#), with a username of [root](#) and a password of [12345](#). The default database for all statements will be the [test](#) database.

The following options are typically used (a full list of options is available in the API documentation for [Section 26.2.3.3.15](#), “[ConnectionString](#)”):

- **Server**: The name or network address of the instance of MySQL to which to connect. The default is [localhost](#). Aliases include [host](#), [Data Source](#), [DataSource](#), [Address](#), [Addr](#) and [Network Address](#).
- **Uid**: The MySQL user account to use when connecting. Aliases include [User Id](#), [Username](#) and [User name](#).
- **Pwd**: The password for the MySQL account being used. Alias [Password](#) can also be used.
- **Database**: The default database that all statements are applied to. Default is [mysql](#). Alias [Initial Catalog](#) can also be used.
- **Port**: The port MySQL is using to listen for connections. Default is [3306](#). Specify [-1](#) for this value to use a named-pipe connection.

26.2.5.1.3. Opening a Connection

Once you have created a connection string it can be used to open a connection to the MySQL server.

The following code is used to create a [MySQLConnection](#) object, assign the connection string, and open the connection.

Visual Basic Example

```
Dim conn As New MySql.Data.MySqlClient.MySqlConnection
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test;"

Try
    conn.ConnectionString = myConnectionString
    conn.Open()

Catch ex As MySql.Data.MySqlClient.MySqlException
    MessageBox.Show(ex.Message)
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection();
    conn.ConnectionString = myConnectionString;
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}
```

You can also pass the connection string to the constructor of the [MySQLConnection](#) class:

Visual Basic Example

```
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    MessageBox.Show(ex.Message)
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}
```

Once the connection is open it can be used by the other Connector/.NET classes to communicate with the MySQL server.

26.2.5.1.4. Handling Connection Errors

Because connecting to an external server is unpredictable, it is important to add error handling to your .NET application. When there is an error connecting, the `MySQLConnection` class will return a `MySqlException` object. This object has two properties that are of interest when handling errors:

- **Message**: A message that describes the current exception.
- **Number**: The MySQL error number.

When handling errors, you can your application's response based on the error number. The two most common error numbers when connecting are as follows:

- **0**: Cannot connect to server.
- **1045**: Invalid username and/or password.

The following code shows how to adapt the application's response based on the actual error:

Visual Basic Example

```
Dim myConnectionString as String
myConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    Select Case ex.Number
        Case 0
            MessageBox.Show("Cannot connect to server. Contact administrator")
        Case 1045
            MessageBox.Show("Invalid username/password, please try again")
    End Select
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    switch (ex.Number)
    {
        case 0:
            MessageBox.Show("Cannot connect to server. Contact administrator");
        case 1045:
            MessageBox.Show("Invalid username/password, please try again");
    }
}
```

Important

Note that if you are using multilanguage databases you must specify the character set in the connection string. If you do

not specify the character set, the connection defaults to the `latin1` charset. You can specify the character set as part of the connection string, for example:

```
MySQLConnection myConnection = new MySqlConnection("server=127.0.0.1;uid=root;" +
"pwd=12345;database=test;Charset=latin1;");
```

26.2.5.2. Using the Connector/NET with Prepared Statements

26.2.5.2.1. Introduction

As of MySQL 4.1, it is possible to use prepared statements with Connector/NET. Use of prepared statements can provide significant performance improvements on queries that are executed more than once.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

26.2.5.2.2. Preparing Statements in Connector/NET

To prepare a statement, create a command object and set the `.CommandText` property to your query.

After entering your statement, call the `.Prepare` method of the `MySQLCommand` object. After the statement is prepared, add parameters for each of the dynamic elements in the query.

After you enter your query and enter parameters, execute the statement using the `.ExecuteNonQuery()`, `.ExecuteScalar()`, or `.ExecuteReader` methods.

For subsequent executions, you need only modify the values of the parameters and call the execute method again, there is no need to set the `.CommandText` property or redefine the parameters.

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = strConnection

Try
    conn.Open()
    cmd.Connection = conn

    cmd.CommandText = "INSERT INTO myTable VALUES(NULL, ?number, ?text)"
    cmd.Prepare()

    cmd.Parameters.Add("?number", 1)
    cmd.Parameters.Add("?text", "One")

    For i = 1 To 1000
        cmd.Parameters["?number"].Value = i
        cmd.Parameters["?text"].Value = "A string value"

        cmd.ExecuteNonQuery()
    Next
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.E
End Try
```

C# Example

```
MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;

conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = strConnection;

try
{
    conn.Open();
```

```

cmd.Connection = conn;
cmd.CommandText = "INSERT INTO myTable VALUES(NULL, ?number, ?text)";
cmd.Prepare();

cmd.Parameters.Add("?number", 1);
cmd.Parameters.Add("?text", "One");

for (int i=1; i <= 1000; i++)
{
    cmd.Parameters["?number"].Value = i;
    cmd.Parameters["?text"].Value = "A string value";

    cmd.ExecuteNonQuery();
}
}
catch (MySQL.Data.MySqlClient.MySQLException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

26.2.5.3. Accessing Stored Procedures with Connector/NET

26.2.5.3.1. Introduction

With the release of MySQL version 5 the MySQL server now supports stored procedures with the SQL 2003 stored procedure syntax.

A stored procedure is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored procedure instead.

Stored procedures can be particularly useful in situations such as the following:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
- When security is paramount. Banks, for example, use stored procedures for all common operations. This provides a consistent and secure environment, and procedures can ensure that each operation is properly logged. In such a setup, applications and users would not get any access to the database tables directly, but can only execute specific stored procedures.

Connector/NET supports the calling of stored procedures through the `MySQLCommand` object. Data can be passed in and out of a MySQL stored procedure through use of the `MySQLCommand.Parameters` collection.

Note

When you call a stored procedure, the command object makes an additional `SELECT` call to determine the parameters of the stored procedure. You must ensure that the user calling the procedure has the `SELECT` privilege on the `mysql.proc` table to enable them to verify the parameters. Failure to do this will result in an error when calling the procedure.

This section will not provide in-depth information on creating Stored Procedures. For such information, please refer to <http://dev.mysql.com/doc/mysql/en/stored-procedures.html>.

A sample application demonstrating how to use stored procedures with Connector/NET can be found in the `Samples` directory of your Connector/NET installation.

26.2.5.3.2. Creating Stored Procedures from Connector/NET

Stored procedures in MySQL can be created using a variety of tools. First, stored procedures can be created using the `mysql` command-line client. Second, stored procedures can be created using the `MySQL Query Browser` GUI client. Finally, stored procedures can be created using the `.ExecuteNonQuery` method of the `MySQLCommand` object:

Visual Basic Example

```

Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

```

```

Try
    conn.Open()
    cmd.Connection = conn

    cmd.CommandText = "CREATE PROCEDURE add_emp(" _
        & "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno INT) " _
        & "BEGIN INSERT INTO emp(first_name, last_name, birthdate) " _
        & "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT_ID(); END"

    cmd.ExecuteNonQuery()
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.E
End Try

```

C# Example

```

MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "CREATE PROCEDURE add_emp(" +
        "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno INT) " +
        "BEGIN INSERT INTO emp(first_name, last_name, birthdate) " +
        "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT_ID(); END";

    cmd.ExecuteNonQuery();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

It should be noted that, unlike the command-line and GUI clients, you are not required to specify a special delimiter when creating stored procedures in Connector/NET.

26.2.5.3.3. Calling a Stored Procedure from Connector/NET

To call a stored procedure using Connector/NET, create a [MySqlCommand](#) object and pass the stored procedure name as the [.CommandText](#) property. Set the [.CommandType](#) property to [CommandType.StoredProcedure](#).

After the stored procedure is named, create one [MySqlCommand](#) parameter for every parameter in the stored procedure. [IN](#) parameters are defined with the parameter name and the object containing the value, [OUT](#) parameters are defined with the parameter name and the datatype that is expected to be returned. All parameters need the parameter direction defined.

After defining parameters, call the stored procedure by using the [MySqlCommand.ExecuteNonQuery\(\)](#) method:

Visual Basic Example

```

Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()
    cmd.Connection = conn

    cmd.CommandText = "add_emp"
    cmd.CommandType = CommandType.StoredProcedure

    cmd.Parameters.Add("?lname", 'Jones')
    cmd.Parameters["?lname"].Direction = ParameterDirection.Input

    cmd.Parameters.Add("?fname", 'Tom')
    cmd.Parameters["?fname"].Direction = ParameterDirection.Input

```

```

cmd.Parameters.Add("?bday", #12/13/1977 2:17:36 PM#)
cmd.Parameters["?bday"].Direction = ParameterDirection.Input

cmd.Parameters.Add("?empno", MySqlDbType.Int32)
cmd.Parameters["?empno"].Direction = ParameterDirection.Output

cmd.ExecuteNonQuery()

MessageBox.Show(cmd.Parameters["?empno"].Value)
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

```

C# Example

```

MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;

conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "add_emp";
    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.Add("?lname", "Jones");
    cmd.Parameters["?lname"].Direction = ParameterDirection.Input;

    cmd.Parameters.Add("?fname", "Tom");
    cmd.Parameters["?fname"].Direction = ParameterDirection.Input;

    cmd.Parameters.Add("?bday", DateTime.Parse("12/13/1977 2:17:36 PM"));
    cmd.Parameters["?bday"].Direction = ParameterDirection.Input;

    cmd.Parameters.Add("?empno", MySqlDbType.Int32);
    cmd.Parameters["?empno"].Direction = ParameterDirection.Output;

    cmd.ExecuteNonQuery();

    MessageBox.Show(cmd.Parameters["?empno"].Value);
}
catch (MySQL.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

Once the stored procedure is called, the values of output parameters can be retrieved by using the `.Value` property of the `MySQLConnector.Parameters` collection.

26.2.5.4. Handling BLOB Data With Connector/NET

26.2.5.4.1. Introduction

One common use for MySQL is the storage of binary data in **BLOB** columns. MySQL supports four different BLOB datatypes: **TINYBLOB**, **BLOB**, **MEDIUMBLOB**, and **LONGBLOB**.

Data stored in a BLOB column can be accessed using Connector/NET and manipulated using client-side code. There are no special requirements for using Connector/NET with BLOB data.

Simple code examples will be presented within this section, and a full sample application can be found in the [Samples](#) directory of the Connector/NET installation.

26.2.5.4.2. Preparing the MySQL Server

The first step in using MySQL with BLOB data is to configure the server. Let's start by creating a table to be accessed. In my file tables, I usually have four columns: an **AUTO_INCREMENT** column of appropriate size (**UNSIGNED SMALLINT**) to serve as a primary key to identify the file, a **VARCHAR** column that stores the filename, an **UNSIGNED MEDIUMINT** column that stores the size of the file, and a **MEDIUMBLOB** column that stores the file itself. For this example, I will use the following table definition:

```
CREATE TABLE file(
file_id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
file_name VARCHAR(64) NOT NULL,
file_size MEDIUMINT UNSIGNED NOT NULL,
file MEDIUMBLOB NOT NULL);
```

After creating a table, you may need to modify the `max_allowed_packet` system variable. This variable determines how large of a packet (i.e. a single row) can be sent to the MySQL server. By default, the server will only accept a maximum size of 1 meg from our client application. If you do not intend to exceed 1 meg, this should be fine. If you do intend to exceed 1 meg in your file transfers, this number has to be increased.

The `max_allowed_packet` option can be modified using MySQL Administrator's Startup Variables screen. Adjust the Maximum allowed option in the Memory section of the Networking tab to an appropriate setting. After adjusting the value, click the **APPLY CHANGES** button and restart the server using the [Service Control](#) screen of MySQL Administrator. You can also adjust this value directly in the `my.cnf` file (add a line that reads `max_allowed_packet=xxM`), or use the `SET max_allowed_packet=xxM;` syntax from within MySQL.

Try to be conservative when setting `max_allowed_packet`, as transfers of BLOB data can take some time to complete. Try to set a value that will be adequate for your intended use and increase the value if necessary.

26.2.5.4.3. Writing a File to the Database

To write a file to a database we need to convert the file to a byte array, then use the byte array as a parameter to an [INSERT](#) query.

The following code opens a file using a `FileStream` object, reads it into a byte array, and inserts it into the `file` table:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

Dim SQL As String

Dim FileSize As UInt32
Dim rawData() As Byte
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    fs = New FileStream("c:\image.png", FileMode.Open, FileAccess.Read)
    FileSize = fs.Length

    rawData = New Byte(FileSize) {}
    fs.Read(rawData, 0, FileSize)
    fs.Close()

    conn.Open()

    SQL = "INSERT INTO file VALUES(NULL, ?FileName, ?FileSize, ?File)"

    cmd.Connection = conn
    cmd.CommandText = SQL
    cmd.Parameters.Add("?FileName", strFileName)
    cmd.Parameters.Add("?FileSize", FileSize)
    cmd.Parameters.Add("?File", rawData)

    cmd.ExecuteNonQuery()

    MessageBox.Show("File Inserted into database successfully!", _
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk)

    conn.Close()
Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", _
        MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;

conn = new MySQL.Data.MySqlClient.MySqlConnection();
```



```

cmd = new MySql.Data.MySqlClient.MySqlCommand();
string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    fs = new FileStream(@"c:\image.png", FileMode.Open, FileAccess.Read);
    FileSize = fs.Length;

    rawData = new byte[FileSize];
    fs.Read(rawData, 0, FileSize);
    fs.Close();

    conn.Open();

    SQL = "INSERT INTO file VALUES(NULL, ?FileName, ?FileSize, ?File)";

    cmd.Connection = conn;
    cmd.CommandText = SQL;
    cmd.Parameters.Add("?FileName", strFileName);
    cmd.Parameters.Add("?FileSize", FileSize);
    cmd.Parameters.Add("?File", rawData);

    cmd.ExecuteNonQuery();

    MessageBox.Show("File Inserted into database successfully!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

The [Read](#) method of the [FileStream](#) object is used to load the file into a byte array which is sized according to the [Length](#) property of the [FileStream](#) object.

After assigning the byte array as a parameter of the [MySqlCommand](#) object, the [ExecuteNonQuery](#) method is called and the BLOB is inserted into the [file](#) table.

26.2.5.4.4. Reading a BLOB from the Database to a File on Disk

Once a file is loaded into the [file](#) table, we can use the [MySqlDataReader](#) class to retrieve it.

The following code retrieves a row from the [file](#) table, then loads the data into a [FileStream](#) object to be written to disk:

Visual Basic Example

```

Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myData As MySqlDataReader
Dim SQL As String
Dim rawData() As Byte
Dim FileSize As UInt32
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" & _
    & "uid=root;" & _
    & "pwd=12345;" & _
    & "database=test"

SQL = "SELECT file_name, file_size, file FROM file"

Try
    conn.Open()

    cmd.Connection = conn
    cmd.CommandText = SQL

    myData = cmd.ExecuteReader

    If Not myData.HasRows Then Throw New Exception("There are no BLOBs to save")

    myData.Read()

```

```

        FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"))
        rawData = New Byte(FileSize) {}

        myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSize)

        fs = New FileStream("C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write)
        fs.Write(rawData, 0, FileSize)
        fs.Close()

        MessageBox.Show("File successfully written to disk!", "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk)

        myData.Close()
        conn.Close()
    Catch ex As Exception
        MessageBox.Show("There was an error: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Try

```

C# Example

```

MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;
MySQL.Data.MySqlClient.MySqlDataReader myData;

conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();

string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

SQL = "SELECT file_name, file_size, file FROM file";

try
{
    conn.Open();

    cmd.Connection = conn;
    cmd.CommandText = SQL;

    myData = cmd.ExecuteReader();

    if (! myData.HasRows)
        throw new Exception("There are no BLOBs to save");

    myData.Read();

    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"));
    rawData = new byte[FileSize];

    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSize);

    fs = new FileStream(@"C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write);
    fs.Write(rawData, 0, FileSize);
    fs.Close();

    MessageBox.Show("File successfully written to disk!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    myData.Close();
    conn.Close();
}
catch (MySQL.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

After connecting, the contents of the **file** table are loaded into a **MySqlDataReader** object. The **GetBytes** method of the **MySqlDataReader** is used to load the BLOB into a byte array, which is then written to disk using a **FileStream** object.

The **GetOrdinal** method of the **MySqlDataReader** can be used to determine the integer index of a named column. Use of the **GetOrdinal** method prevents errors if the column order of the **SELECT** query is changed.

26.2.5.5. Using Connector/NET with Crystal Reports

26.2.5.5.1. Introduction

Crystal Reports is a common tool used by Windows application developers to perform reporting and document generation. In this section we will show how to use Crystal Reports XI with MySQL and Connector/NET.

26.2.5.5.2. Creating a Data Source

When creating a report in Crystal Reports there are two options for accessing the MySQL data while designing your report.

The first option is to use Connector/ODBC as an ADO data source when designing your report. You will be able to browse your database and choose tables and fields using drag and drop to build your report. The disadvantage of this approach is that additional work must be performed within your application to produce a dataset that matches the one expected by your report.

The second option is to create a dataset in VB.NET and save it as XML. This XML file can then be used to design a report. This works quite well when displaying the report in your application, but is less versatile at design time because you must choose all relevant columns when creating the dataset. If you forget a column you must re-create the dataset before the column can be added to the report.

The following code can be used to create a dataset from a query and write it to disk:

Visual Basic Example

```
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"

Try
    conn.Open()
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myData.WriteXml("C:\dataset.xml", XmlWriteMode.WriteSchema)
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
DataSet myData = new DataSet();
MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;
MySQL.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();
myAdapter = new MySQL.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
        "country.name, country.population, country.continent " +
        "FROM country, city ORDER BY country.continent, country.name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myData.WriteXml(@"C:\dataset.xml", XmlWriteMode.WriteSchema);
}
catch (MySQL.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

The resulting XML file can be used as an ADO.NET XML datasource when designing your report.

If you choose to design your reports using Connector/ODBC, it can be downloaded from dev.mysql.com.

26.2.5.5.3. Creating the Report

For most purposes the Standard Report wizard should help with the initial creation of a report. To start the wizard, open Crystal Reports and choose the New > Standard Report option from the File menu.

The wizard will first prompt you for a data source. If you are using Connector/ODBC as your data source, use the OLEDB provider for ODBC option from the OLE DB (ADO) tree instead of the ODBC (RDO) tree when choosing a data source. If using a saved dataset, choose the ADO.NET (XML) option and browse to your saved dataset.

The remainder of the report creation process is done automatically by the wizard.

After the report is created, choose the Report Options... entry of the File menu. Un-check the Save Data With Report option. This prevents saved data from interfering with the loading of data within our application.

26.2.5.5.4. Displaying the Report

To display a report we first populate a dataset with the data needed for the report, then load the report and bind it to the dataset. Finally we pass the report to the crViewer control for display to the user.

The following references are needed in a project that displays a report:

- CrystalDecisions.CrystalReports.Engine
- CrystalDecisions.ReportSource
- CrystalDecisions.Shared
- CrystalDecisions.Windows.Forms

The following code assumes that you created your report using a dataset saved using the code shown in [Section 26.2.5.5.2, “Creating a Data Source”](#), and have a crViewer control on your form named `myViewer`.

Visual Basic Example

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = _
    "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()

    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myReport.Load(".\world_report.rpt")
    myReport.SetDataSource(myData)
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```

using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
        "country.name, country.population, country.continent " +
        "FROM country, city ORDER BY country.continent, country.name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myReport.Load(@"..\world_report.rpt");
    myReport.SetDataSource(myData);
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

A new dataset is generated using the same query used to generate the previously saved dataset. Once the dataset is filled, a ReportDocument is used to load the report file and bind it to the dataset. The ReportDocument is then passed as the ReportSource of the crViewer.

This same approach is taken when a report is created from a single table using Connector/ODBC. The dataset replaces the table used in the report and the report is displayed properly.

When a report is created from multiple tables using Connector/ODBC, a dataset with multiple tables must be created in our application. This allows each table in the report data source to be replaced with a report in the dataset.

We populate a dataset with multiple tables by providing multiple **SELECT** statements in our MySqlCommand object. These **SELECT** statements are based on the SQL query shown in Crystal Reports in the Database menu's Show SQL Query option. Assume the following query:

```

SELECT `country`.`Name`, `country`.`Continent`, `country`.`Population`, `city`.`Name`, `city`.`Population`
FROM `world`.`country` `country` LEFT OUTER JOIN `world`.`city` `city` ON `country`.`Code` = `city`.`CountryCode`
ORDER BY `country`.`Continent`, `country`.`Name`, `city`.`Name`

```

This query is converted to two **SELECT** queries and displayed with the following code:

Visual Basic Example

```

Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"

Try
    conn.Open()
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER BY countrycode, name;" _
        & "SELECT name, population, code, continent FROM country ORDER BY continent, name"
    cmd.Connection = conn

```

```

myAdapter.SelectCommand = cmd
myAdapter.Fill(myData)

myReport.Load(".\world_report.rpt")
myReport.Database.Tables(0).SetDataSource(myData.Tables(0))
myReport.Database.Tables(1).SetDataSource(myData.Tables(1))
myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

```

C# Example

```

using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER " +
        "BY countrycode, name; SELECT name, population, code, continent FROM " +
        "country ORDER BY continent, name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myReport.Load(@".\world_report.rpt");
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0));
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1));
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

It is important to order the **SELECT** queries in alphabetical order, as this is the order the report will expect its source tables to be in. One **SetDataSource** statement is needed for each table in the report.

This approach can cause performance problems because Crystal Reports must bind the tables together on the client-side, which will be slower than using a pre-saved dataset.

26.2.5.6. Handling Date and Time Information in Connector/NET

26.2.5.6.1. Introduction

MySQL and the .NET languages handle date and time information differently, with MySQL allowing dates that cannot be represented by a .NET data type, such as '0000-00-00 00:00:00'. These differences can cause problems if not properly handled.

In this section we will demonstrate how to properly handle date and time information when using Connector/NET.

26.2.5.6.2. Problems when Using Invalid Dates

The differences in date handling can cause problems for developers who use invalid dates. Invalid MySQL dates cannot be loaded into native .NET **DateTime** objects, including **NULL** dates.

Because of this issue, .NET **DataSet** objects cannot be populated by the **Fill** method of the **MySqlDataAdapter** class as invalid dates will cause a **System.ArgumentOutOfRangeException** exception to occur.

26.2.5.6.3. Restricting Invalid Dates

The best solution to the date problem is to restrict users from entering invalid dates. This can be done on either the client or the server side.

Restricting invalid dates on the client side is as simple as always using the .NET `DateTime` class to handle dates. The `DateTime` class will only allow valid dates, ensuring that the values in your database are also valid. The disadvantage of this is that it is not useful in a mixed environment where .NET and non .NET code are used to manipulate the database, as each application must perform its own date validation.

Users of MySQL 5.0.2 and higher can use the new `traditional` SQL mode to restrict invalid date values. For information on using the `traditional` SQL mode, see [Section 5.2.6, “SQL Modes”](#).

26.2.5.6.4. Handling Invalid Dates

Although it is strongly recommended that you avoid the use of invalid dates within your .NET application, it is possible to use invalid dates by means of the `MySqlDateTime` datatype.

The `MySqlDateTime` datatype supports the same date values that are supported by the MySQL server. The default behavior of Connector/NET is to return a .NET `DateTime` object for valid date values, and return an error for invalid dates. This default can be modified to cause Connector/NET to return `MySqlDateTime` objects for invalid dates.

To instruct Connector/NET to return a `MySqlDateTime` object for invalid dates, add the following line to your connection string:

```
Allow Zero Datetime=True
```

Please note that the use of the `MySqlDateTime` class can still be problematic. The following are some known issues:

1. Data binding for invalid dates can still cause errors (zero dates like 0000-00-00 do not seem to have this problem).
2. The `ToString` method return a date formatted in the standard MySQL format (for example, `2005-02-23 08:50:25`). This differs from the `ToString` behavior of the .NET `DateTime` class.
3. The `MySqlDateTime` class supports NULL dates, while the .NET `DateTime` class does not. This can cause errors when trying to convert a `MySqlDateTime` to a `DateTime` if you do not check for NULL first.

Because of the known issues, the best recommendation is still to use only valid dates in your application.

26.2.5.6.5. Handling NULL Dates

The .NET `DateTime` datatype cannot handle `NULL` values. As such, when assigning values from a query to a `DateTime` variable, you must first check whether the value is in fact `NULL`.

When using a `MySqlDataReader`, use the `.IsDBNull` method to check whether a value is `NULL` before making the assignment:

Visual Basic Example

```
If Not myReader.IsDBNull(myReader.GetOrdinal("mytime")) Then
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"))
Else
    myTime = DateTime.MinValue
End If
```

C# Example

```
if (! myReader.IsDBNull(myReader.GetOrdinal("mytime")))
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"));
else
    myTime = DateTime.MinValue;
```

`NULL` values will work in a dataset and can be bound to form controls without special handling.

26.2.6. Connector/NET Support

The developers of Connector/NET greatly value the input of our users in the software development process. If you find Connector/NET lacking some feature important to you, or if you discover a bug and need to file a bug report, please use the instructions in [Section 1.8](#), “How to Report Bugs or Problems”.

26.2.6.1. Connector/NET Community Support

- Community support for Connector/NET can be found through the forums at <http://forums.mysql.com>.
- Community support for Connector/NET can also be found through the mailing lists at <http://lists.mysql.com>.
- Paid support is available from MySQL AB. Additional information is available at <http://www.mysql.com/support/>.

26.2.6.2. How to report Connector/NET Problems or Bugs

If you encounter difficulties or problems with Connector/NET, contact the Connector/NET community [Section 26.2.6.1](#), “Connector/NET Community Support”.

You should first try to execute the same SQL statements and commands from the `mysql` client program or from `admindemo`. This helps you determine whether the error is in Connector/NET or MySQL.

If reporting a problem, you should ideally include the following information with the email:

- Operating system and version
- Connector/NET version
- MySQL server version
- Copies of error messages or other unexpected output
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

If you believe the problem to be a bug, then you must report the bug through <http://bugs.mysql.com/>.

26.2.6.3. Connector/NET Change History

The Connector/NET Change History (Changelog) is located with the main Changelog for MySQL. See [Section C.3](#), “Connector/NET Change History”.

26.3. MySQL Visual Studio Plugin

The MySQL Visual Studio Plugin is a DDEX provider; a plug-in for Visual Studio 2005 that allows developers to maintain database structures, and supports built-in data-driven application development tools.

The current version of the MySQL Visual Studio Plugin includes only database maintenance tools. Data-driven application development tools are not supported.

The MySQL DDEX Provider operates as a standard extension to the Visual Studio Data Designer functionality available through the Server Explorer menu of Visual Studio 2005, and enables developers to create database objects and data within a MySQL database.

The MySQL Visual Studio Plugin is designed to work with MySQL version 5.0, but is also compatible with MySQL 4.1.1 and provides limited compatibility with MySQL 5.1.

26.3.1. Installing the MySQL Visual Studio Plugin

The MySQL Visual Studio Plugin requires one of Visual Studio 2005 Standard, Professional or Team Developer Edition to be installed.

Other editions of Visual Studio 2005 are not supported.

Note

Starting with Connector/NET 5.1.2, the Visual Studio Plugin is included in the installation. If you have installed Connector/NET 5.1.2, then you do not need to separately install the Visual Studio Plugin.

Here is the list of components that should already be installed before starting the installation of the MySQL Visual Studio Plugin:

- Visual Studio 2005 Standard, Professional or Team Developer Edition.
- MySQL Server 4.1.1 or later (either installed on the same machine, or a separate server).
- MySQL Connector/NET 5.0.

Note

When installing Connector/NET you must ensure that the connector is installed into the Global Assembly Cache (GAC). The Connector/NET installer handles this for you automatically, but in a custom installation the option may have been disabled.

The user used to connect to the MySQL server must have the following privileges to use the functionality provided by the MySQL Visual Studio Plugin:

- The **SELECT** privilege for the **INFORMATION_SCHEMA** database.
- The **EXECUTE** privilege for the **SHOW CREATE TABLE** statement.
- The **SELECT** privilege for the **mysql.proc** table (required for operations with stored procedures and functions).
- The **SELECT** privilege for the **mysql.func** table (required for operations with User Defined Functions (UDF)).
- The **EXECUTE** privilege for the **SHOW ENGINE STATUS** statement (required for retrieving extended error information).
- Appropriate privileges for performed operations (e.g. the **SELECT** privilege is required to browse data from a table etc.).

The MySQL Visual Studio Plugin is delivered as a MSI package that can be used to install, uninstall or reinstall the Provider. If you are not using Windows XP or Windows Server 2003 you upgrade the Windows Installer system to the latest version (see <http://support.microsoft.com/default.aspx?scid=kb;EN-US;292539> for details).

The MSI-package is named **MySQL.VisualStudio.msi**. To install the MySQL Visual Studio Plugin, right click on the MSI file and select **INSTALL**. The installation process is as follow:

1. The standard Welcome dialog is opened. Click Next to continue installation.
2. The License agreement (GNU GPL) window is opened. Accept the agreement and click NEXT to continue.
3. The destination folder choice dialog is opened. Here you can point out the folder where the MySQL Visual Studio Plugin will be installed. The default destination folder is **%ProgramFilesDir%\MySQL\MySQL DDEX Data Provider**, where **%ProgramFilesDir%** is the Program Files folder of the installation machine. After choosing the destination folder, click NEXT to continue.
4. The installer will ask to confirm that installation. Click Install to start installation process.
5. The installation will now take place. At the end of this step the Visual Studio command table is rebuilt (this process may take several minutes).
6. Once installation is complete, click FINISH to end the installation process.

To uninstall the MySQL Visual Studio Plugin, you can use either Add/Remove Programs component of the Control Panel or the same

MSI-package. Choose the **REMOVE** option, and the Provider will be uninstalled automatically.

To repair the Provider, right click the MSI-package and choose the **REPAIR** option. The MySQL Visual Studio Plugin will be repaired automatically.

The installation package includes the following files:

- **MySQL.VisualStudio.dll** — the MySQL DDEX Provider assembly.
- **MySQL.Data.dll** — the assembly containing the MySQL Connector .NET which is used by the Provider.
- **MySQL.VisualStudio.dll.config** — the configuration file for the MySQL Visual Studio Plugin. This file contains default values for the provider GUI layout.

Note

Do not remove this file before the first use of the Provider.

- **Register.reg** — the file with registry entries that can be used to register the MySQL DDEX Provider in the case of the manual installation.
- **Install.js** — the script used to register the Connector .NET as an ADO.NET data provider in the machine.config file.
- **Release notes.doc** — the document with release notes.

To install the Provider manually, copy all files of the installation package in a desired folder, then set the full path to the Provider assembly as a value of the CodeBase entry. For example:

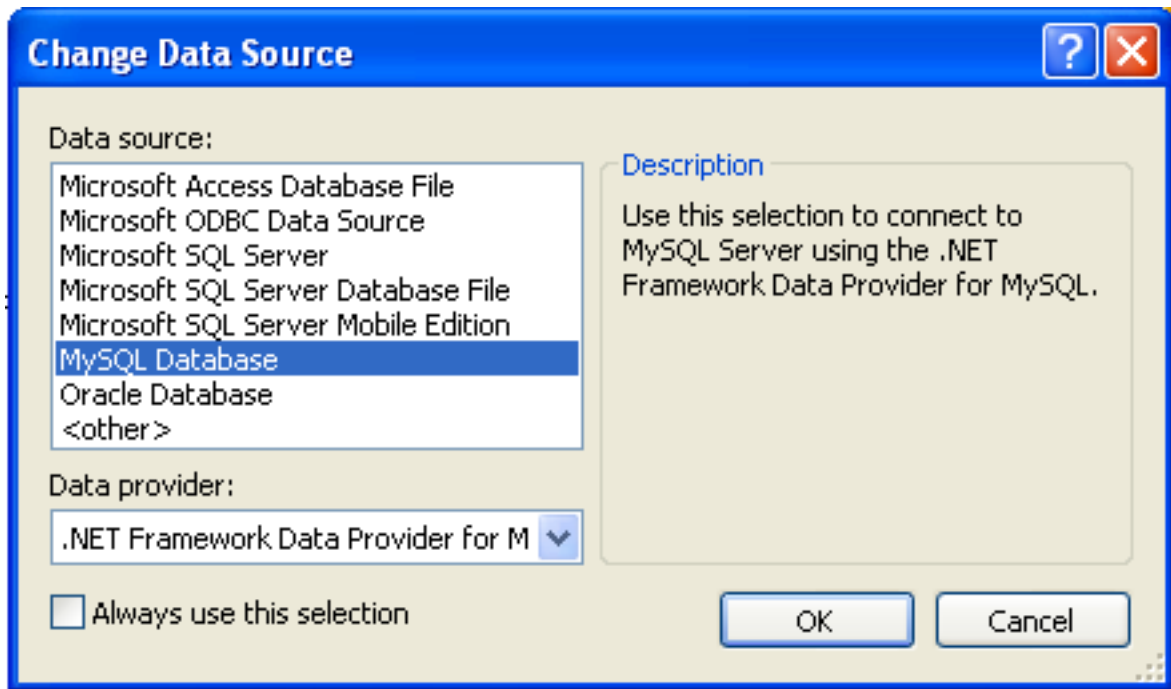
```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\Packages\{79A115C9-B133-4891-9E7B-242509DAD272}]@="MySQL.Data.VisualStudio"
"InprocServer32"="C:\\WINNT\\system32\\mscorlib.dll"
"Class"="MySQL.Data.VisualStudio.MySqlDataProviderPackage"
"CodeBase"="C:\\MySQLDdexProvider\\MySQL.VisualStudio.dll"
```

Then import information from the Register.reg file to the registry by clicking of the file. At the confirmation dialog choose Yes. Next you must run the command `devenv.exe /setup` within a Command Prompt to rebuild the Visual Studio command table.

26.3.2. Creating a connection to the MySQL server

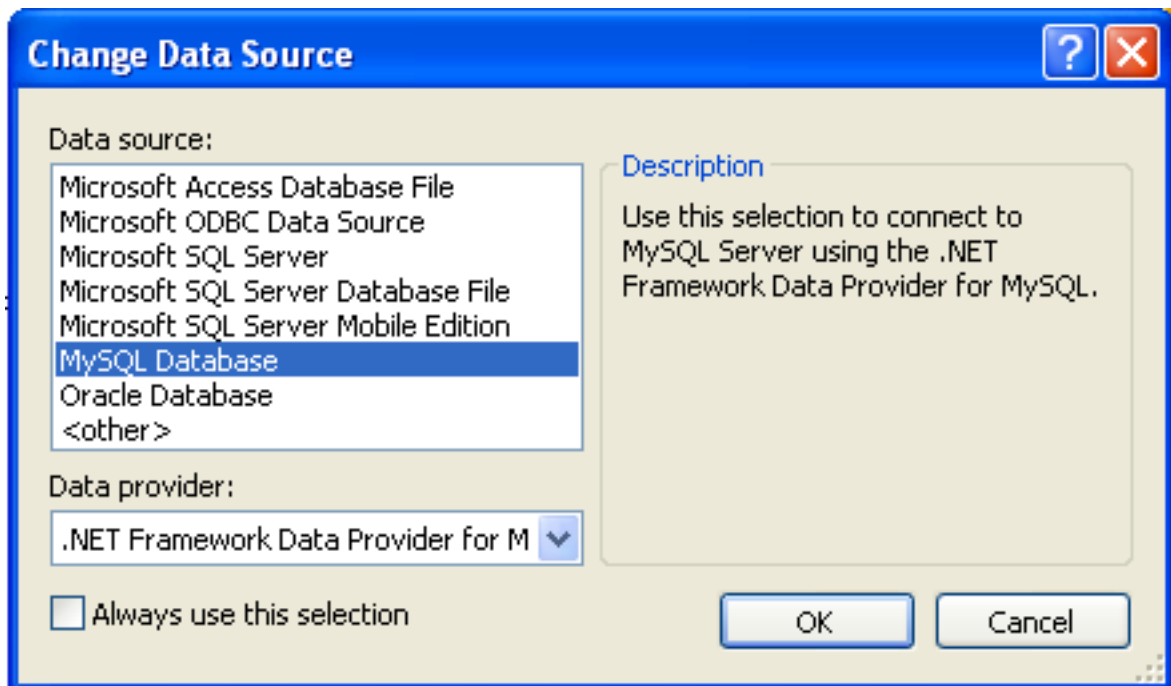
Once the MySQL Visual Studio Plugin is installed, you can use it to create, modify and delete connections to MySQL databases. To create a connection with a MySQL database, perform the following steps:

1. Start Visual Studio 2005 and open Server Explorer window by choosing the **SERVER EXPLORER** option from the **VIEW** menu.
2. Right click on the **DATA CONNECTIONS** node and choose the **ADD CONNECTION** button.
3. The Add Connection dialog is opened. Press the **CHANGE** button to choose MySQL Database as a data source.
4. Change Data Source dialog is opened. Choose MySQL Database in the list of data sources (or the **other** option, if MySQL Database is absent), and then choose **.NET FRAMEWORK DATA PROVIDER FOR MYSQL** in the combo box of data providers.



Press OK to confirm your choice.

5. Enter the connection settings: the server host name (for example, localhost if the MySQL server is installed on the local machine), the user name, the password, and the default database schema. Note that you must specify the default schema name to open the connection.



6. You can also set the port to connect with the MySQL server by pressing the ADVANCED button. To test a connection with the MySQL server, set the server host name, the user name, and the password, and press the TEST CONNECTION button. If the test fails, check the connection values that you have supplied are correct and that the corresponding user and privileges have been configured on the MySQL server.

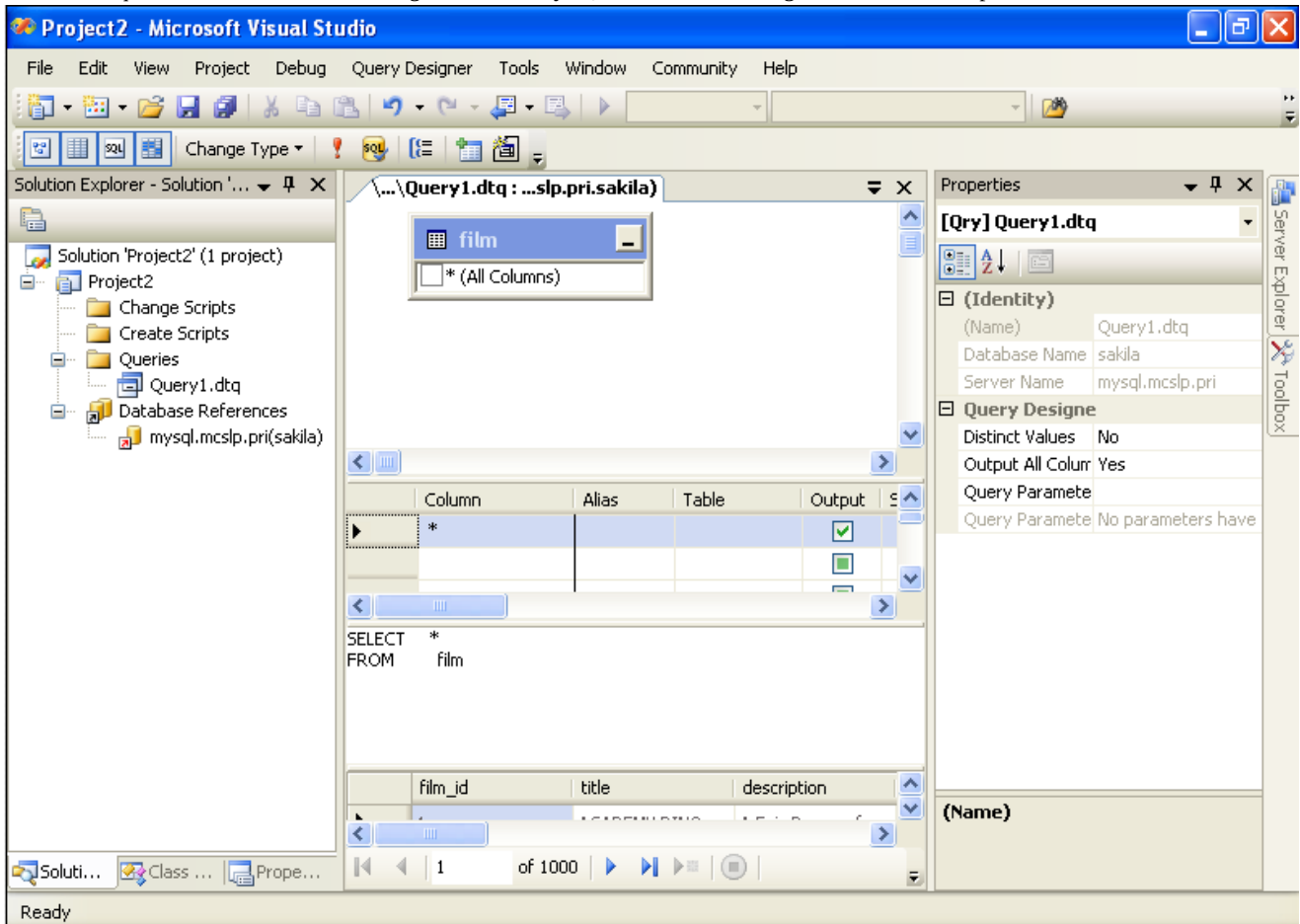
- After you set all settings and test the connection, press OK. The newly created connection is displayed in Server Explorer. Now you can work with the MySQL server through standard Server Explorer interface.

After a connection is successfully established, all the connection settings are saved. When you next open Visual Studio, the connection to the MySQL server will appear within Server Explorer so that you can re-establish a connection to the MySQL server.

To modify and delete a connection, use the **SERVER EXPLORER** context menu for the corresponding node. You can modify any of the settings just by overwriting the existing values with new ones. Note that a connection should be modified or deleted only if no active editor for its objects is opened. Otherwise your data could be lost.

26.3.3. Using the MySQL Visual Studio Plugin

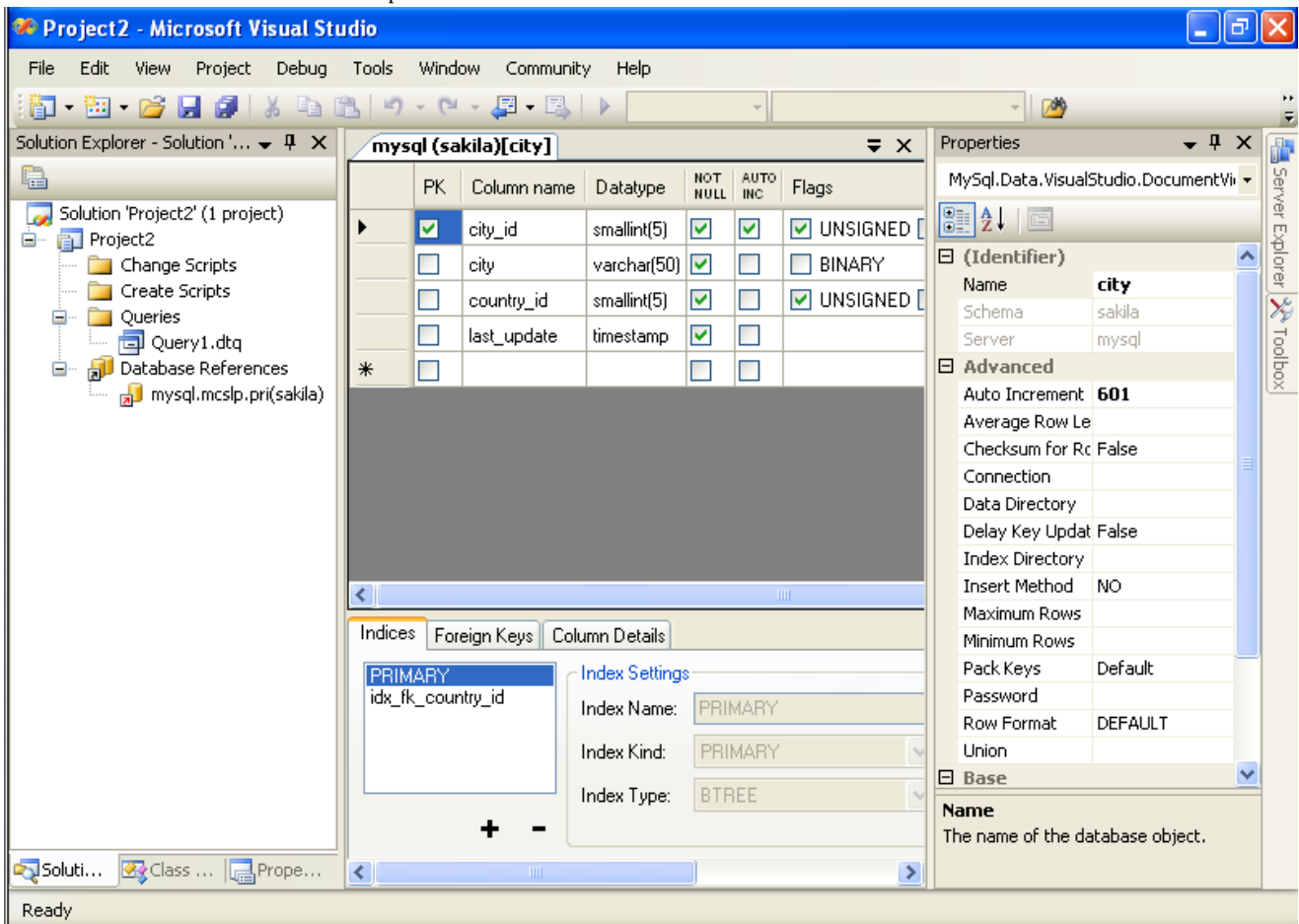
To work with a MySQL server using the MySQL Visual Studio Plugin, open the Visual Studio 2005, open the **SERVER EXPLORER**, and select the required connection. The working area of the MySQL Visual Studio Plugin consists of three parts.



- Database objects (tables, views, stored routines, triggers, and user defined functions) are displayed in the Server Explorer tree. Here you can choose an object and edit its properties and definition.
- Properties of a selected database object are displayed in the **PROPERTIES** panel. Certain properties can be edited directly within this window.
- The editor panel provides direct access to the SQL statement and definition of specific objects. For example, the SQL statements within a stored procedure definition are shown and edited within this panel.

26.3.3.1. Editing Tables

The Table Editor can be accessed through a mouse action on table-type node of Server Explorer. To create a new table, right click on the **TABLES** node (under the connection node) and choose the **CREATE TABLE** command from a context menu. To modify an existing table, double click on a node of the table you wish to modify, or right click on this node and choose the **ALTER TABLE** command from a context menu. Either of the commands opens the Table Editor.



The MySQL Visual Studio Plugin Table Editor is implemented in a similar fashion to the standard Query Browser Table Editor, but with minor differences.

The Table Editor consists of the following parts:

- Columns Editor — for column creation, modification and deletion.
- Indexes tab — for table/column index management.
- Foreign Keys tab — for configuration of foreign keys.
- Column Details tab — used to set advanced column options.
- Properties window — used to set table properties.

To save changes you have made in the Table Editor, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. Before changes are saved, a confirmation dialog will be displayed to confirm that you want to update the corresponding object within the MySQL database.

26.3.3.1.1. Column Editor

You can use the Column Editor to set or change the name, data type, default value and other properties of a table column. To set the

properties of an individual column, select the column using the mouse. Alternatively, you can move through the grid using **Tab** and **Shift+Tab** keys.

- To set or change the name, data type, default value and comment of a column, select the appropriate cell and edit the desired value.
- To set or unset flag-type column properties (i.e., primary key, **NOT NULL**, auto incremented, flags), check or uncheck the corresponding checkboxes. Note that the available column flags will depend on the columns data type.
- To reorder columns, index columns or foreign key columns in the Column Editor, select the whole column you wish to reorder by clicking on the selector column at the left of the column grid. Then move the column by using **Ctrl+Up** (to move the column up) and **Ctrl+Down** (to move the column down) keys.
- To delete a column, select it by clicking on the selector column at the left of the column grid, then press the **Delete** button on a keyboard.

26.3.3.1.2. Indexes tab

Index management is performed via the Indexes tab.

- To add an index, press the + button and set the properties in the **INDEX SETTINGS** groupbox at the right. You can set the index name, index kind, index type and a set of index columns.
- To remove an index, select the index from the list and press the - button.
- To change index settings, select the index from the list; detailed information about the index is displayed in the **INDEX SETTINGS** panel.

You cannot change a table column to an index column using drag and drop. Instead, you can add new index columns to a table and set their table columns by using the embedded editor within the Indexes tab

26.3.3.1.3. Foreign Keys tab

Foreign Key management is performed via the Foreign Keys tab.

- To add a foreign key, press the + button and set properties in the **FOREIGN KEYS SETTINGS** panel. You can set the foreign key name, referenced table name, foreign key columns and actions on update and delete.
- To remove a foreign key, select the foreign key and press the - button.
- To change foreign key settings, select the foreign key and use the **FOREIGN KEYS SETTINGS** panel to edit the properties.
- When a foreign key is changed, the MySQL Visual Studio Plugin generates two queries: the first query drops the changed keys and the second one recreates the new values. The reason for such a behavior is to avoid the [Bug#8377](#) and [Bug#8919](#).

Note

If changed values are for some reason inconsistent and cause the second query to fail, all affected foreign keys will be dropped. If this is the case, the MySQL Visual Studio Plugin will mark them as new in the Table Editor, and you will have to recreate them later. But if you close the Table Editor without saving, these foreign keys will be lost.

26.3.3.1.4. Column Details tab

The Column Details tab can be used to set column options. Besides the main column properties that are presented in the Column Editor, in the Column Details tab you can set two additional properties options: the character set and the collation sequence.

26.3.3.1.5. Table Properties window

There is no separate tab for table options and advanced options. All table options can be browsed and changed using the **PROPERTIES** window of Visual Studio 2005.

The following table properties can be set:

- **AUTO INCREMENT**
- **AVERAGE ROW LENGTH**
- **CHARACTER SET**
- **CHECKSUM FOR ROWS**
- **COLLATION**
- **COMMENT**
- **CONNECTION**
- **DATA DIRECTORY**
- **DELAY KEY UPDATES**
- **ENGINE**
- **INDEX DIRECTORY**
- **INSERT METHOD**
- **MAXIMUM ROWS**
- **MINIMUM ROWS**
- **NAME**
- **PACK KEYS**
- **PASSWORD**
- **ROW FORMAT**
- **UNION**

Some of these properties can have arbitrary text values, others accept values from a predefined set.

The properties **SCHEMA** and **SERVER** are read only.

26.3.3.2. Editing Table Data

The Table Data Editor, allows a user to browse, create and edit data of tables. The Table Data Editor is implemented as a simple data grid with auto generated columns.

To access the Table Data Editor, right click on a node representing the table or view in Server Explorer. From the nodes context menu, choose the BROWSE or EEDIT DATA command. For tables and updatable views, this command opens the Table Data Editor in edit mode. For non-updatable views, this command opens the Table Data Editor in read-only mode.

When in the edit mode, you can modify table data by modifying the displayed table contents directly. To add a row, set desired values in the last row of the grid. To modify values, set new values in appropriate cells. To delete a row, select it by clicking on the selector column at the left of the grid, then press the **DELETE** button.

To save changes you have made in the Table Data Editor, use either **SAVE** or **SAVE ALL** buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

26.3.3.3. Editing Views

To create a new view, right click the Views node under the connection node in Server Explorer. From the nodes context menu, choose the CREATE VIEW command. This command opens the SQL Editor.

To modify an existing view, double click on a node of the view you wish to modify, or right click on this node and choose the [ALTER VIEW](#) command from a context menu. Either of the commands opens the SQL Editor.

To create or alter the view definition using SQL Editor, type the appropriate SQL statement in the SQL Editor.

Note

You should enter only the defining statement itself, without the [CREATE VIEW AS](#) preface.

All other view properties can be set in the **PROPERTIES** window. These properties are:

- **ALGORITHM**
- **CHECK OPTION**
- **DEFINER**
- **NAME**
- **SECURITY TYPE**

Some of these properties can have arbitrary text values, others accept values from a predefined set.

The properties **IS UPDATABLE**, **SCHEMA** and **SERVER** are readonly.

To save changes you have made, use either **SAVE** or **SAVE ALL** buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

26.3.3.4. Editing Stored Procedures and Functions

To create a new stored procedure, right click the **Stored Procedures** node under the connection node in Server Explorer. From the nodes context menu, choose the [CREATE ROUTINE](#) command. This command opens the SQL Editor.

To create a new stored function, right click the **FUNCTIONS** node under the connection node in Server Explorer. From the node's context menu, choose the [CREATE ROUTINE](#) command.

To modify an existing stored routine (procedure or function), double click on a node of the routine you wish to modify, or right click on this node and choose the [ALTER ROUTINE](#) command from a context menu. Either of the commands opens the SQL Editor.

To create or alter the routine definition using SQL Editor, type this definition in the SQL Editor using standard SQL.

All other routine properties can be set in the **PROPERTIES** window. These properties are:

- **Comment**
- **Data Access**
- **Definer**
- **Is Deterministic**
- **Security Type**

Some of these properties can have arbitrary text values, others accept values only from a predefined set.

Also you can set all the options directly in the SQL Editor, using the standard [CREATE PROCEDURE](#) or [CREATE FUNCTION](#) statement. However, it is recommended to use the **PROPERTIES** window instead.

Note

You should never add the [CREATE](#) preface to the routine definition.

The properties **NAME**, **SCHEMA** and **SERVER** in the **PROPERTIES** window are read-only. Set or change the procedure name in the SQL editor.

To save changes you have made, use either **SAVE** or **SAVE ALL** buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database..

26.3.3.5. Editing Triggers

To create a new trigger, right click on a node of a table for which you wish to add a trigger. From the node's context menu, choose the **CREATE TRIGGER** command. This command opens the SQL Editor.

To modify an existing trigger, double click on a node of the trigger you wish to modify, or right click on this node and choose the **ALTER TRIGGER** command from a context menu. Either of the commands opens the SQL Editor.

To create or alter the trigger definition using SQL Editor, type the trigger statement in the SQL Editor using standard SQL.

Note

You should enter only the trigger statement, that is the part of the **CREATE TRIGGER** query that is placed after the **FOR EACH ROW** clause.

All other trigger properties are set in the **PROPERTIES** window. These properties are:

- **DEFINER**
- **EVENT MANIPULATION**
- **NAME**
- **TIMING**

Some of these properties can have arbitrary text values, others accept values only from a predefined set.

The properties **EVENT TABLE**, **SCHEMA** and **SERVER** in the **PROPERTIES** window are read-only.

To save changes you have made, use either **SAVE** or **SAVE ALL** buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

26.3.3.6. Editing User Defined Functions (UDF)

To create a new User Defined Function (UDF), right click the UDFs node under the connection node in Server Explorer. From the node's context menu, choose the **CREATE UDF** command. This command opens the UDF Editor.

To modify an existing UDF, double click on a node of the UDF you wish to modify, or right click on this node and choose the **Alter UDF** command from a context menu. Either of the commands opens the UDF Editor.

The UDF editor allows you to set the following properties through the properties panel:

- **NAME**
- **SO-NAME (DLL NAME)**
- **RETURN TYPE**
- **IS AGGREGATE**

The property **Server** in the **PROPERTIES** window is read-only.

To save changes you have made, use either **SAVE** or **SAVE ALL** buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

26.3.3.7. Dropping database objects

Tables, views, stored routines, triggers, and UDFs can be dropped with the appropriate DROP command from its context menu: DROP TABLE, DROP VIEW, DROP ROUTINE, DROP TRIGGER, DROP UDF.

You will be asked to confirm the execution of the corresponding drop query in a confirmation dialog.

Dropping of multiple objects is not supported.

26.3.3.8. Cloning database objects

Tables, views, stored procedures and functions can be cloned with the appropriate CLONE command from its context menu: CLONE TABLE, CLONE VIEW, CLONE ROUTINE. The clone commands open the corresponding editor for a new object: the **TABLE EDITOR** for cloning a table and the SQL Editor for cloning a view or a routine.

To save the cloned object, use either **SAVE** or **SAVE ALL** buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

26.3.4. Visual Studio Plugin Support

If you have a comment, or if you discover a bug, please, use our MySQL bug tracking system (<http://bugs.mysql.com>) to report problem or add your suggestion.

26.3.4.1. Visual Studio Plugin FAQ

Questions

- [27.3.4.1.1](#): When creating a connection, typing the connection details causes the connection window to immediately close.

Questions and Answers

27.3.4.1.1: When creating a connection, typing the connection details causes the connection window to immediately close.

There are known issues with versions of Connector/NET earlier than 5.0.2. Connector/NET 1.0.x is known not to work. If you have any of these versions installed, or have previously upgraded from an earlier version, uninstall Connector/NET completely and then install Connector/NET 5.0.2.

26.4. MySQL Connector/J

MySQL provides connectivity for client applications developed in the Java programming language via a JDBC driver, which is called MySQL Connector/J.

MySQL Connector/J is a JDBC-3.0 Type 4 driver, which means that is pure Java, implements version 3.0 of the JDBC specification, and communicates directly with the MySQL server using the MySQL protocol.

Although JDBC is useful by itself, we would hope that if you are not familiar with JDBC that after reading the first few sections of this manual, that you would avoid using naked JDBC for all but the most trivial problems and consider using one of the popular persistence frameworks such as [Hibernate](#), [Spring's JDBC templates](#) or [Ibatis SQL Maps](#) to do the majority of repetitive work and heavier lifting that is sometimes required with JDBC.

This section is not designed to be a complete JDBC tutorial. If you need more information about using JDBC you might be interested in the following online tutorials that are more in-depth than the information presented here:

- [JDBC Basics](#) — A tutorial from Sun covering beginner topics in JDBC
- [JDBC Short Course](#) — A more in-depth tutorial from Sun and JGuru

Key topics:

- For help with connection strings, connection options setting up your connection through JDBC, see [Section 26.4.4.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”](#).
- For tips on using Connector/J and JDBC with generic J2EE toolkits, see [Section 26.4.5.2, “Using Connector/J with J2EE and Other Java Frameworks”](#).
- Developers using the Tomcat server platform, see [Section 26.4.5.2.2, “Using Connector/J with Tomcat”](#).
- Developers using JBoss, see [Section 26.4.5.2.3, “Using Connector/J with JBoss”](#).
- Developers using Spring, see [Section 26.4.5.2.4, “Using Connector/J with Spring”](#).

MySQL Enterprise

MySQL Enterprise subscribers will find more information about using JDBC with MySQL in the Knowledge Base articles about [JDBC](#). Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

26.4.1. Connector/J Versions

There are currently four versions of MySQL Connector/J available:

- Connector/J 5.1 is current in alpha status. It provides compatibility with all the functionality of MySQL, including 4.1, 5.0, 5.1 and the 6.0 alpha release featuring the new Falcon storage engine. Connector/J 5.1 provides ease of development features, including auto-registration with the Driver Manager, standardized validity checks, categorized SQLExceptions, support for the JDBC-4.0 XML processing, per connection client information, [NCHAR](#), [NVARCHAR](#) and [NCLOB](#) types. This release also includes all bug fixes up to and including Connector/J 5.0.6.
- Connector/J 5.0 provides support for all the functionality offered by Connector/J 3.1 and includes distributed transaction (XA) support.
- Connector/J 3.1 was designed for connectivity to MySQL 4.1 and MySQL 5.0 servers and provides support for all the functionality in MySQL 5.0 except distributed transaction (XA) support.
- Connector/J 3.0 provides core functionality and was designed with connectivity to MySQL 3.x or MySQL 4.1 servers, although it will provide basic compatibility with later versions of MySQL. Connector/J 3.0 does not support server-side prepared statements, and does not support any of the features in versions of MySQL later than 4.1.

The current recommended version for Connector/J is 5.0. This guide covers all three connector versions, with specific notes given where a setting applies to a specific option.

26.4.1.1. Java Versions Supported

MySQL Connector/J supports Java-2 JVMs, including:

- JDK 1.2.x (only for Connector/J 3.1.x or earlier)
- JDK 1.3.x
- JDK 1.4.x
- JDK 1.5.x

If you are building Connector/J from source using the source distribution (see [Section 26.4.2.4, “Installing from the Development Source Tree”](#)) then you must use JDK 1.4.x or newer to compile the Connector package.

MySQL Connector/J does not support JDK-1.1.x or JDK-1.0.x.

Because of the implementation of [java.sql.Savepoint](#), Connector/J 3.1.0 and newer will not run on JDKs older than 1.4 unless the class verifier is turned off (by setting the [-Xverify:none](#) option to the Java runtime). This is because the class verifier will try to

load the class definition for `java.sql.Savepoint` even though it is not accessed by the driver unless you actually use savepoint functionality.

Caching functionality provided by Connector/J 3.1.0 or newer is also not available on JVMs older than 1.4.x, as it relies on `java.util.LinkedHashMap` which was first available in JDK-1.4.0.

26.4.2. Connector/J Installation

You can install the Connector/J package using two methods, using either the binary or source distribution. The binary distribution provides the easiest methods for installation; the source distribution enables you to customize your installation further. With either solution, you must manually add the Connector/J location to your Java `CLASSPATH`.

26.4.2.1. Installing Connector/J from a Binary Distribution

The easiest method of installation is to use the binary distribution of the Connector/J package. The binary distribution is available either as a Tar/Gzip or Zip file which you must extract to a suitable location and then optionally make the information about the package available by changing your `CLASSPATH` (see [Section 26.4.2.2, “Installing the Driver and Configuring the CLASSPATH”](#)).

MySQL Connector/J is distributed as a .zip or .tar.gz archive containing the sources, the class files, and the JAR archive named `mysql-connector-java-[version]-bin.jar`, and starting with Connector/J 3.1.8 a debug build of the driver in a file named `mysql-connector-java-[version]-bin-g.jar`.

Starting with Connector/J 3.1.9, the .class files that constitute the JAR files are only included as part of the driver JAR file.

You should not use the debug build of the driver unless instructed to do so when reporting a problem or a bug to MySQL AB, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

You will need to use the appropriate graphical or command-line utility to extract the distribution (for example, WinZip for the .zip archive, and `tar` for the .tar.gz archive). Because there are potentially long filenames in the distribution, we use the GNU tar archive format. You will need to use GNU tar (or an application that understands the GNU tar archive format) to unpack the .tar.gz variant of the distribution.

26.4.2.2. Installing the Driver and Configuring the CLASSPATH

Once you have extracted the distribution archive, you can install the driver by placing `mysql-connector-java-[version]-bin.jar` in your classpath, either by adding the full path to it to your `CLASSPATH` environment variable, or by directly specifying it with the command line switch `-cp` when starting your JVM.

If you are going to use the driver with the JDBC DriverManager, you would use `com.mysql.jdbc.Driver` as the class that implements `java.sql.Driver`.

You can set the `CLASSPATH` environment variable under UNIX, Linux or Mac OS X either locally for a user within their `.profile`, `.login` or other login file. You can also set it globally by editing the global `/etc/profile` file.

For example, under a C shell (csh, tcsh) you would add the Connector/J driver to your `CLASSPATH` using the following:

```
shell> setenv CLASSPATH /path/mysql-connector-java-[ver]-bin.jar:$CLASSPATH
```

Or with a Bourne-compatible shell (sh, ksh, bash):

```
export set CLASSPATH=/path/mysql-connector-java-[ver]-bin.jar:$CLASSPATH
```

Within Windows 2000, Windows XP and Windows Server 2003, you must set the environment variable through the System control panel.

If you want to use MySQL Connector/J with an application server such as Tomcat or JBoss, you will have to read your vendor's documentation for more information on how to configure third-party class libraries, as most application servers ignore the `CLASSPATH` environment variable. For configuration examples for some J2EE application servers, see [Section 26.4.5.2, “Using Connector/J with J2EE and Other Java Frameworks”](#). However, the authoritative source for JDBC connection pool configuration information for your particular application server is the documentation for that application server.

If you are developing servlets or JSPs, and your application server is J2EE-compliant, you can put the driver's .jar file in the WEB-

INF/lib subdirectory of your webapp, as this is a standard location for third party class libraries in J2EE web applications.

You can also use the `MysqlDataSource` or `MysqlConnectionPoolDataSource` classes in the `com.mysql.jdbc.jdbc2.optional` package, if your J2EE application server supports or requires them. Starting with Connector/J 5.0.0, the `javax.sql.XADataSource` interface is implemented via the `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` class, which supports XA distributed transactions when used in combination with MySQL server version 5.0.

The various `MysqlDataSource` classes support the following parameters (through standard set mutators):

- `user`
- `password`
- `serverName` (see the previous section about fail-over hosts)
- `databaseName`
- `port`

26.4.2.3. Upgrading from an Older Version

MySQL AB tries to keep the upgrade process as easy as possible, however as is the case with any software, sometimes changes need to be made in new versions to support new features, improve existing functionality, or comply with new standards.

This section has information about what users who are upgrading from one version of Connector/J to another (or to a new version of the MySQL server, with respect to JDBC functionality) should be aware of.

26.4.2.3.1. Upgrading from MySQL Connector/J 3.0 to 3.1

Connector/J 3.1 is designed to be backward-compatible with Connector/J 3.0 as much as possible. Major changes are isolated to new functionality exposed in MySQL-4.1 and newer, which includes Unicode character sets, server-side prepared statements, `SQLState` codes returned in error messages by the server and various performance enhancements that can be enabled or disabled via configuration properties.

- **Unicode Character Sets** — See the next section, as well as [Chapter 9, Character Set Support](#), for information on this new feature of MySQL. If you have something misconfigured, it will usually show up as an error with a message similar to `Illegal mix of collations`.
- **Server-side Prepared Statements** — Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer).

Starting with version 3.1.7, the driver scans SQL you are preparing via all variants of `Connection.prepareStatement()` to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this feature by passing `emulateUnsupportedPstmts=false` in your JDBC URL.

If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the connection property `useServerPrepStmts=false`

- **Datetimes** with all-zero components (`0000-00-00 . . .`) — These values can not be represented reliably in Java. Connector/J 3.0.x always converted them to `NULL` when being read from a `ResultSet`.

Connector/J 3.1 throws an exception by default when these values are encountered as this is the most correct behavior according to the JDBC and SQL standards. This behavior can be modified using the `zeroDateTimeBehavior` configuration property. The allowable values are:

- `exception` (the default), which throws an `SQLException` with an `SQLState` of `S1009`.
- `convertToNull`, which returns `NULL` instead of the date.
- `round`, which rounds the date to the nearest closest value which is `0001-01-01`.

Starting with Connector/J 3.1.7, `ResultSet.getString()` can be decoupled from this behavior via `noDatetimeStringSync=true` (the default value is `false`) so that you can get retrieve the unaltered all-zero value as a String. It should be noted that this also precludes using any time zone conversions, therefore the driver will not allow you to enable `noDatetimeStringSync` and `useTimezone` at the same time.

- **New SQLState Codes** — Connector/J 3.1 uses SQL:1999 SQLState codes returned by the MySQL server (if supported), which are different from the legacy X/Open state codes that Connector/J 3.0 uses. If connected to a MySQL server older than MySQL-4.1.0 (the oldest version to return SQLStates as part of the error code), the driver will use a built-in mapping. You can revert to the old mapping by using the configuration property `useSqlStateCodes=false`.
- **`ResultSet.getString()`** — Calling `ResultSet.getString()` on a BLOB column will now return the address of the `byte[]` array that represents it, instead of a String representation of the BLOB. BLOBs have no character set, so they can't be converted to `java.lang.Strings` without data loss or corruption.

To store strings in MySQL with LOB behavior, use one of the TEXT types, which the driver will treat as a `java.sql.Clob`.

- **Debug builds** — Starting with Connector/J 3.1.8 a debug build of the driver in a file named `mysql-connector-java-[version]-bin-g.jar` is shipped alongside the normal binary jar file that is named `mysql-connector-java-[version]-bin.jar`.

Starting with Connector/J 3.1.9, we don't ship the .class files unbundled, they are only available in the JAR archives that ship with the driver.

You should not use the debug build of the driver unless instructed to do so when reporting a problem or bug to MySQL AB, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

26.4.2.3.2. JDBC-Specific Issues When Upgrading to MySQL Server 4.1 or Newer

- *Using the UTF-8 Character Encoding* - Prior to MySQL server version 4.1, the UTF-8 character encoding was not supported by the server, however the JDBC driver could use it, allowing storage of multiple character sets in latin1 tables on the server.

Starting with MySQL-4.1, this functionality is deprecated. If you have applications that rely on this functionality, and can not upgrade them to use the official Unicode character support in MySQL server version 4.1 or newer, you should add the following property to your connection URL:

`useOldUTF8Behavior=true`

- *Server-side Prepared Statements* - Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer). If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the following connection property:

`useServerPrepStmts=false`

26.4.2.4. Installing from the Development Source Tree

Caution

You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL Connector/J up and running on your system, you should use a standard release distribution.

To install MySQL Connector/J from the development source tree, make sure that you have the following prerequisites:

- Subversion, to check out the sources from our repository (available from <http://subversion.tigris.org/>).
- Apache Ant version 1.6 or newer (available from <http://ant.apache.org/>).
- JDK-1.4.2 or later. Although MySQL Connector/J can be installed on older JDKs, to compile it from source you must have at least

JDK-1.4.2.

The Subversion source code repository for MySQL Connector/J is located at <http://svn.mysql.com/svnpublic/connector-j>. In general, you should not check out the entire repository because it contains every branch and tag for MySQL Connector/J and is quite large.

To check out and compile a specific branch of MySQL Connector/J, follow these steps:

1. At the time of this writing, there are three active branches of Connector/J: `branch_3_0`, `branch_3_1` and `branch_5_0`. Check out the latest code from the branch that you want with the following command (replacing `[major]` and `[minor]` with appropriate version numbers):

```
shell> svn co »
http://svn.mysql.com/svnpublic/connector-j/branches/branch_[major]_[minor]/connector-j
```

This creates a `connector-j` subdirectory in the current directory that contains the latest sources for the requested branch.

2. Change location to the `connector-j` directory to make it your current working directory:

```
shell> cd connector-j
```

3. Issue the following command to compile the driver and create a `.jar` file suitable for installation:

```
shell> ant dist
```

This creates a `build` directory in the current directory, where all build output will go. A directory is created in the `build` directory that includes the version number of the sources you are building from. This directory contains the sources, compiled `.class` files, and a `.jar` file suitable for deployment. For other possible targets, including ones that will create a fully packaged distribution, issue the following command:

```
shell> ant --projecthelp
```

4. A newly created `.jar` file containing the JDBC driver will be placed in the directory `build/mysql-connector-java-[version]`.

Install the newly created JDBC driver as you would a binary `.jar` file that you download from MySQL by following the instructions in [Section 26.4.2.2, “Installing the Driver and Configuring the CLASSPATH”](#).

26.4.3. Connector/J Examples

Examples of using Connector/J are located throughout this document, this section provides a summary and links to these examples.

- [Example 26.1, “Obtaining a connection from the DriverManager”](#)
- [Example 26.2, “Using java.sql.Statement to execute a SELECT query”](#)
- [Example 26.3, “Stored Procedures”](#)
- [Example 26.4, “Using Connection.prepareStatement\(\)”](#)
- [Example 26.5, “Registering output parameters”](#)
- [Example 26.6, “Setting CallableStatement input parameters”](#)
- [Example 26.7, “Retrieving results and output parameter values”](#)
- [Example 26.8, “Retrieving AUTO_INCREMENT column values using Statement.getGeneratedKeys\(\)”](#)
- [Example 26.9, “Retrieving AUTO_INCREMENT column values using SELECT LAST_INSERT_ID\(\)”](#)
- [Example 26.10, “Retrieving AUTO_INCREMENT column values in Updatable ResultSets”](#)

- [Example 26.11, “Using a connection pool with a J2EE application server”](#)
- [Example 26.12, “Example of transaction with retry logic”](#)

26.4.4. Connector/J (JDBC) Reference

This section of the manual contains reference material for MySQL Connector/J, some of which is automatically generated during the Connector/J build process.

26.4.4.1. Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J

The name of the class that implements `java.sql.Driver` in MySQL Connector/J is `com.mysql.jdbc.Driver`. The `org.gjt.mm.mysql.Driver` class name is also usable to remain backward-compatible with MM.MySQL. You should use this class name when registering the driver, or when otherwise configuring software to use MySQL Connector/J.

The JDBC URL format for MySQL Connector/J is as follows, with items in square brackets ([,]) being optional:

```
jdbc:mysql://[host][, failoverhost...][:port]/[database] »  
[?propertyName1[=propertyValue1]&propertyName2[=propertyValue2]...
```

If the hostname is not specified, it defaults to 127.0.0.1. If the port is not specified, it defaults to 3306, the default port number for MySQL servers.

```
jdbc:mysql://[host:port],[host:port].../[database] »  
[?propertyName1[=propertyValue1]&propertyName2[=propertyValue2]...
```

If the database is not specified, the connection will be made with no default database. In this case, you will need to either call the `setCatalog()` method on the Connection instance or fully-specify table names using the database name (i.e. `SELECT dbname.tablename.colname FROM dbname.tablename...`) in your SQL. Not specifying the database to use upon connection is generally only useful when building tools that work with multiple databases, such as GUI database managers.

MySQL Connector/J has fail-over support. This allows the driver to fail-over to any number of slave hosts and still perform read-only queries. Fail-over only happens when the connection is in an `autoCommit(true)` state, because fail-over can not happen reliably when a transaction is in progress. Most application servers and connection pools set `autoCommit` to `true` at the end of every transaction/connection use.

The fail-over functionality has the following behavior:

- If the URL property `autoReconnect` is false: Failover only happens at connection initialization, and fallback occurs when the driver determines that the first host has become available again.
- If the URL property `autoReconnect` is true: Failover happens when the driver determines that the connection has failed (before *every* query), and falls back to the first host when it determines that the host has become available again (after `queriesBeforeRetryMaster` queries have been issued).

In either case, whenever you are connected to a "failed-over" server, the connection will be set to read-only state, so queries that would modify data will have exceptions thrown (the query will **never** be processed by the MySQL server).

Configuration properties define how Connector/J will make a connection to a MySQL server. Unless otherwise noted, properties can be set for a `DataSource` object or for a `Connection` object.

Configuration Properties can be set in one of the following ways:

- Using the `set*()` methods on MySQL implementations of `java.sql.DataSource` (which is the preferred method when using implementations of `java.sql.DataSource`):
 - `com.mysql.jdbc.jdbc2.optional.MysqlDataSource`
 - `com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource`

- As a key/value pair in the `java.util.Properties` instance passed to `DriverManager.getConnection()` or `Driver.connect()`
- As a JDBC URL parameter in the URL given to `java.sql.DriverManager.getConnection()`, `java.sql.Driver.connect()` or the MySQL implementations of the `javax.sql.DataSource.setURL()` method.

Note

If the mechanism you use to configure a JDBC URL is XML-based, you will need to use the XML character literal `&` to separate configuration parameters, as the ampersand is a reserved character for XML.

The properties are listed in the following tables.

Connection/Authentication.

Property Name	Definition	Default Value	Since Version
user	The user to connect as		all versions
password	The password to use when connecting		all versions
socketFactory	The name of the class that the driver should use for creating socket connections to the server. This class must implement the interface 'com.mysql.jdbc.SocketFactory' and have public no-args constructor.	com.mysql.jdbc.StandardSocketFactory	3.0.3
connectTimeout	Timeout for socket connect (in milliseconds), with 0 being no timeout. Only works on JDK-1.4 or newer. Defaults to '0'.	0	3.0.1
socketTimeout	Timeout on network socket operations (0, the default means no timeout).	0	3.0.1
useConfigs	Load the comma-delimited list of configuration properties before parsing the URL or applying user-specified properties. These configurations are explained in the 'Configurations' of the documentation.		3.1.5
interactiveClient	Set the CLIENT_INTERACTIVE flag, which tells MySQL to timeout connections based on INTERACTIVE_TIMEOUT instead of WAIT_TIMEOUT	false	3.1.0
localSocketAddress	Hostname or IP address given to explicitly configure the interface that the driver will bind the client side of the TCP/IP connection to when connecting.		5.0.5
propertiesTransform	An implementation of com.mysql.jdbc.ConnectionPropertiesTransform that the driver will use to modify URL properties passed to the driver before attempting a connection		3.1.4
useCompression	Use zlib compression when communicating with the server (true/false)? Defaults to 'false'.	false	3.0.17

Networking.

Property Name	Definition	Default Value	Since Version
tcpKeepAlive	If connecting using TCP/IP, should the driver set SO_KEEPALIVE?	true	5.0.7
tcpNoDelay	If connecting using TCP/IP, should the driver set SO_TCP_NODELAY (disabling the Nagle Algorithm)?	true	5.0.7
tcpRcvBuf	If connecting using TCP/IP, should the driver set SO_RCV_BUF to the given value? The default value of '0', means use the platform default value for this property)	0	5.0.7
tcpSndBuf	If connecting using TCP/IP, should the driver set SO_SND_BUF	0	5.0.7

	to the given value? The default value of '0', means use the platform default value for this property)		
tcpTrafficClass	If connecting using TCP/IP, should the driver set traffic class or type-of-service fields ?See the documentation for <code>java.net.Socket.setTrafficClass()</code> for more information.	0	5.0.7

High Availability and Clustering.

Property Name	Definition	Default Value	Since Version
autoReconnect	Should the driver try to re-establish stale and/or dead connections? If enabled the driver will throw an exception for a queries issued on a stale or dead connection, which belong to the current transaction, but will attempt reconnect before the next query issued on the connection in a new transaction. The use of this feature is not recommended, because it has side effects related to session state and data consistency when applications don't handle <code>SQLExceptions</code> properly, and is only designed to be used when you are unable to configure your application to handle <code>SQLExceptions</code> resulting from dead and stale connections properly. Alternatively, investigate setting the MySQL server variable "wait_timeout" to some high value rather than the default of 8 hours.	false	1.1
autoReconnectForPools	Use a reconnection strategy appropriate for connection pools (defaults to 'false')	false	3.1.3
failOverReadOnly	When failing over in autoReconnect mode, should the connection be set to 'read-only'?	true	3.0.12
maxReconnects	Maximum number of reconnects to attempt if autoReconnect is true, default is '3'.	3	1.1
reconnectAtTxEnd	If autoReconnect is set to true, should the driver attempt reconnects at the end of every transaction?	false	3.0.10
initialTimeout	If autoReconnect is enabled, the initial time to wait between reconnect attempts (in seconds, defaults to '2').	2	1.1
roundRobinLoadBalance	When autoReconnect is enabled, and failoverReadonly is false, should we pick hosts to connect to on a round-robin basis?	false	3.1.2
queriesBeforeRetryMaster	Number of queries to issue before falling back to master when failed over (when using multi-host failover). Whichever condition is met first, 'queriesBeforeRetryMaster' or 'secondsBeforeRetryMaster' will cause an attempt to be made to reconnect to the master. Defaults to 50.	50	3.0.2
secondsBeforeRetryMaster	How long should the driver wait, when failed over, before attempting	30	3.0.2
resourceId	A globally unique name that identifies the resource that this data-source or connection is connected to, used for <code>XAResource.isSameRM()</code> when the driver can't determine this value based on hostnames used in the URL		5.0.1

Security.

Property Name	Definition	Default Value	Since Version
allowMultiQueries	Allow the use of ';' to delimit multiple queries during one statement (true/false), defaults to 'false'	false	3.1.1
useSSL	Use SSL when communicating with the server (true/false), defaults to 'false'	false	3.0.2
requireSSL	Require SSL connection if useSSL=true? (defaults to 'false').	false	3.1.0

allowLoadLocalInfile	Should the driver allow use of 'LOAD DATA LOCAL INFILE...' (defaults to 'true').	true	3.0.3
allowUrlInLocalInfile	Should the driver allow URLs in 'LOAD DATA LOCAL INFILE' statements?	false	3.1.4
clientCertificateKeyStorePassword	Password for the client certificates KeyStore		5.1.0
clientCertificateKeyStoreType	KeyStore type for client certificates (NULL or empty means use default, standard keystore types supported by the JVM are "JKS" and "PKCS12", your environment may have more available depending on what security products are installed and available to the JVM.		5.1.0
clientCertificateKeyStoreUrl	URL to the client certificate KeyStore (if not specified, use defaults)		5.1.0
trustCertificateKeyStorePassword	Password for the trusted root certificates KeyStore		5.1.0
trustCertificateKeyStoreType	KeyStore type for trusted root certificates (NULL or empty means use default, standard keystore types supported by the JVM are "JKS" and "PKCS12", your environment may have more available depending on what security products are installed and available to the JVM.		5.1.0
trustCertificateKeyStoreUrl	URL to the trusted root certificate KeyStore (if not specified, use defaults)		5.1.0
paranoid	Take measures to prevent exposure sensitive information in error messages and clear data structures holding sensitive data when possible? (defaults to 'false')	false	3.0.1

Performance Extensions.

Property Name	Definition	Default Value	Since Version
callableStmtCacheSize	If 'cacheCallableStmts' is enabled, how many callable statements should be cached?	100	3.1.2
metadataCacheSize	The number of queries to cache ResultSetMetadata for if cacheResultSetMetaData is set to 'true' (default 50)	50	3.1.1
prepStmtCacheSize	If prepared statement caching is enabled, how many prepared statements should be cached?	25	3.0.10
prepStmtCacheSqlLimit	If prepared statement caching is enabled, what's the largest SQL the driver will cache the parsing for?	256	3.0.10
alwaysSendSetIsolation	Should the driver always communicate with the database when Connection.setTransactionIsolation() is called? If set to false, the driver will only communicate with the database when the requested transaction isolation is different than the whichever is newer, the last value that was set via Connection.setTransactionIsolation(), or the value that was read from the server when the connection was established.	true	3.1.7
maintainTimeStats	Should the driver maintain various internal timers to enable idle time calculations as well as more verbose error messages when the connection to the server fails? Setting this property to false removes at least two calls to System.currentTimeMillis() per query.	true	3.1.9
useCursorFetch	If connected to MySQL > 5.0.2, and setFetchSize() > 0 on a statement, should that statement use cursor-based fetching to retrieve rows?	false	5.0.0
blobSendChunkSize	Chunk to use when sending BLOB/CLOBs via ServerPreparedStatements	1048576	3.1.9
cacheCallableStmts	Should the driver cache the parsing stage of CallableStatements	false	3.1.2
cachePrepStmts	Should the driver cache the parsing stage of PreparedStatements of	false	3.0.10

	client-side prepared statements, the "check" for suitability of server-side prepared and server-side prepared statements themselves?		
cacheResultSetMetadata	Should the driver cache ResultSetMetaData for Statements and PreparedStatements? (Req. JDK-1.4+, true/false, default 'false')	false	3.1.1
cacheServerConfiguration	Should the driver cache the results of 'SHOW VARIABLES' and 'SHOW COLLATION' on a per-URL basis?	false	3.1.5
defaultFetchSize	The driver will call setFetchSize(n) with this value on all newly-created Statements	0	3.1.9
dontTrackOpenResources	The JDBC specification requires the driver to automatically track and close resources, however if your application doesn't do a good job of explicitly calling close() on statements or result sets, this can cause memory leakage. Setting this property to true relaxes this constraint, and can be more memory efficient for some applications.	false	3.1.7
dynamicCalendars	Should the driver retrieve the default calendar when required, or cache it per connection/session?	false	3.1.5
elideSetAutoCommits	If using MySQL-4.1 or newer, should the driver only issue 'set autocommit=n' queries when the server's state doesn't match the requested state by Connection.setAutoCommit(boolean)?	false	3.1.3
enableQueryTimeouts	When enabled, query timeouts set via Statement.setQueryTimeout() use a shared java.util.Timer instance for scheduling. Even if the timeout doesn't expire before the query is processed, there will be memory used by the TimerTask for the given timeout which won't be reclaimed until the time the timeout would have expired if it hadn't been cancelled by the driver. High-load environments might want to consider disabling this functionality.	true	5.0.6
holdResultsOpenOverStatementClose	Should the driver close result sets on Statement.close() as required by the JDBC specification?	false	3.1.7
largeRowSizeThreshold	What size result set row should the JDBC driver consider "large", and thus use a more memory-efficient way of representing the row internally?	2048	5.1.1
loadBalanceStrategy	If using a load-balanced connection to connect to SQL nodes in a MySQL Cluster/NDB configuration (by using the URL prefix "jdbc:mysql:loadbalance://"), which load balancing algorithm should the driver use: (1) "random" - the driver will pick a random host for each request. This tends to work better than round-robin, as the randomness will somewhat account for spreading loads where requests vary in response time, while round-robin can sometimes lead to overloaded nodes if there are variations in response times across the workload. (2) "bestResponseTime" - the driver will route the request to the host that had the best response time for the previous transaction.	random	5.0.6
locatorFetchBufferSize	If 'emulateLocators' is configured to 'true', what size buffer should be used when fetching BLOB data for getBinaryInputStream?	1048576	3.2.1
rewriteBatchedStatements	Should the driver use multiqueries (irregardless of the setting of "allowMultiQueries") as well as rewriting of prepared statements for INSERT into multi-value inserts when executeBatch() is called? Notice that this has the potential for SQL injection if using plain java.sql.Statements and your code doesn't sanitize input correctly. Notice that for prepared statements, server-side prepared statements can not currently take advantage of this rewrite option, and that if you don't specify stream lengths when using PreparedStatement.set*Stream(), the driver won't be able to determine the optimum number of parameters per batch and you might receive an error from the driver that the resultant packet is too large. Statement.getGeneratedKeys() for these rewritten statements only works when the entire batch includes INSERT statements.	false	3.1.13

useDirectRowUnpack	Use newer result set row unpacking code that skips a copy from network buffers to a MySQL packet instance and instead reads directly into the result set row data buffers.	true	5.1.1
useDynamicCharsetInfo	Should the driver use a per-connection cache of character set information queried from the server when necessary, or use a built-in static mapping that is more efficient, but isn't aware of custom character sets or character sets implemented after the release of the JDBC driver?	true	5.0.6
useFastDateParsing	Use internal String->Date/Time/Timestamp conversion routines to avoid excessive object creation?	true	5.0.5
useFastIntParsing	Use internal String->Integer conversion routines to avoid excessive object creation?	true	3.1.4
useJvmCharsetConverters	Always use the character encoding routines built into the JVM, rather than using lookup tables for single-byte character sets?	false	5.0.1
useLocalSessionState	Should the driver refer to the internal values of autocommit and transaction isolation that are set by Connection.setAutoCommit() and Connection.setTransactionIsolation() and transaction state as maintained by the protocol, rather than querying the database or blindly sending commands to the database for commit() or rollback() method calls?	false	3.1.7
useReadAheadInput	Use newer, optimized non-blocking, buffered input stream when reading from the server?	true	3.1.5

Debugging/Profiling.

Property Name	Definition	Default Value	Since Version
logger	The name of a class that implements "com.mysql.jdbc.log.Log" that will be used to log messages to. (default is "com.mysql.jdbc.log.StandardLogger", which logs to STDERR)	com.mysql.jdbc.log.StandardLogger	3.1.1
gatherPerfMetrics	Should the driver gather performance metrics, and report them via the configured logger every 'reportMetricsIntervalMillis' milliseconds?	false	3.1.2
profileSQL	Trace queries and their execution/fetch times to the configured logger (true/false) defaults to 'false'	false	3.1.0
profileSql	Deprecated, use 'profileSQL' instead. Trace queries and their execution/fetch times on STDERR (true/false) defaults to 'false'		2.0.14
reportMetricsIntervalMillis	If 'gatherPerfMetrics' is enabled, how often should they be logged (in ms)?	30000	3.1.2
maxQuerySizeToLog	Controls the maximum length/size of a query that will get logged when profiling or tracing	2048	3.1.3
packetDebugBufferSize	The maximum number of packets to retain when 'enablePacketDebug' is true	20	3.1.3
slowQueryThresholdMillis	If 'logSlowQueries' is enabled, how long should a query (in ms) before it is logged as 'slow'?	2000	3.1.2
slowQueryThresholdNanos	If 'useNanosForElapsedTime' is set to true, and this property is set to a non-zero value, the driver will use this threshold (in nano-second units) to determine if a query was slow.	0	5.0.7
useUsageAdvisor	Should the driver issue 'usage' warnings advising proper and efficient usage of JDBC and MySQL Connector/J to the log (true/false, defaults to 'false')?	false	3.1.1
autoGenerateTestcaseScript	Should the driver dump the SQL it is executing, including server-side prepared statements to STDERR?	false	3.1.9
clientInfoProvider	The name of a class that implements the	com.mysql.j	5.1.0

	com.mysql.jdbc.JDBC4ClientInfoProvider interface in order to support JDBC-4.0's Connection.getClientInfo() methods	dbc.JDBC4 CommentClientInfoProvider	
dumpMetadataOnColumnNotFound	Should the driver dump the field-level metadata of a result set into the exception message when ResultSet.findColumn() fails?	false	3.1.13
dumpQueriesOnException	Should the driver dump the contents of the query sent to the server in the message for SQLExceptions?	false	3.1.3
enablePacketDebug	When enabled, a ring-buffer of 'packetDebugBufferSize' packets will be kept, and dumped when exceptions are thrown in key areas in the driver's code	false	3.1.3
explainSlowQueries	If 'logSlowQueries' is enabled, should the driver automatically issue an 'EXPLAIN' on the server and send the results to the configured log at a WARN level?	false	3.1.2
includeInnodbStatusInDeadlockExceptions	Include the output of "SHOW ENGINE INNODB STATUS" in exception messages when deadlock exceptions are detected?	false	5.0.7
logSlowQueries	Should queries that take longer than 'slowQueryThresholdMillis' be logged?	false	3.1.2
logXaCommands	Should the driver log XA commands sent by MysqlXaConnection to the server, at the DEBUG level of logging?	false	5.0.5
resultSetSizeThreshold	If the usage advisor is enabled, how many rows should a result set contain before the driver warns that it is suspiciously large?	100	5.0.5
traceProtocol	Should trace-level network protocol be logged?	false	3.1.2
useNanosForElapsedTime	For profiling/debugging functionality that measures elapsed time, should the driver try to use nanoseconds resolution if available (JDK >= 1.5)?	false	5.0.7

Miscellaneous.

Property Name	Definition	Default Value	Since Version
useUnicode	Should the driver use Unicode character encodings when handling strings? Should only be used when the driver can't determine the character set mapping, or you are trying to 'force' the driver to use a character set that MySQL either doesn't natively support (such as UTF-8), true/false, defaults to 'true'	true	1.1g
characterEncoding	If 'useUnicode' is set to true, what character encoding should the driver use when dealing with strings? (defaults is to 'autodetect')		1.1g
characterSetResults	Character set to tell the server to return results as.		3.0.13
connectionCollation	If set, tells the server to use this collation via 'set collation_connection'		3.0.13
useBlobToStoreUTF8OutsideBMP	Tells the driver to treat [MEDIUM/LONG]BLOB columns as [LONG]VARCHAR columns holding text encoded in UTF-8 that has characters outside the BMP (4-byte encodings), which MySQL server can't handle natively.	false	5.1.3
utf8OutsideBmpExcludedColumnNamePattern	When "useBlobToStoreUTF8OutsideBMP" is set to "true", column names matching the given regex will still be treated as BLOBs unless they match the regex specified for "utf8OutsideBmpIncludedColumnNamePattern". The regex must follow the patterns used for the java.util.regex package.		5.1.3
utf8OutsideBmpIncludedColumnNamePattern	Used to specify exclusion rules to "utf8OutsideBmpExcludedColumnNamePattern". The regex must follow the patterns used for the java.util.regex package.		5.1.3
sessionVariables	A comma-separated list of name/value pairs to be sent as SET		3.1.8

	SESSION ... to the server when the driver connects.		
allowNaNAndInf	Should the driver allow NaN or +/- INF values in PreparedStatement.setDouble()?	false	3.1.5
autoClosePstmtStreams	Should the driver automatically call .close() on streams/readers passed as arguments via set*() methods?	false	3.1.12
autoDeserialize	Should the driver automatically detect and de-serialize objects stored in BLOB fields?	false	3.1.5
blobsAreStrings	Should the driver always treat BLOBs as Strings - specifically to work around dubious metadata returned by the server for GROUP BY clauses?	false	5.0.8
capitalizeTypeNames	Capitalize type names in DatabaseMetaData? (usually only useful when using WebObjects, true/false, defaults to 'false')	false	2.0.7
clobCharacterEncoding	The character encoding to use for sending and retrieving TEXT, MEDIUMTEXT and LONGTEXT values instead of the configured connection characterEncoding		5.0.0
clobberStreamingResults	This will cause a 'streaming' ResultSet to be automatically closed, and any outstanding data still streaming from the server to be discarded if another query is executed before all the data has been read from the server.	false	3.0.9
continueBatchOnError	Should the driver continue processing batch commands if one statement fails. The JDBC spec allows either way (defaults to 'true').	true	3.0.3
createDatabaseIfNotExist	Creates the database given in the URL if it doesn't yet exist. Assumes the configured user has permissions to create databases.	false	3.1.9
emptyStringsConvertToZero	Should the driver allow conversions from empty string fields to numeric values of '0'?	true	3.1.8
emulateLocators	Should the driver emulate java.sql.Blobs with locators? With this feature enabled, the driver will delay loading the actual Blob data until the one of the retrieval methods (getInputStream(), getBytes(), and so forth) on the blob data stream has been accessed. For this to work, you must use a column alias with the value of the column to the actual name of the Blob. The feature also has the following restrictions: The SELECT that created the result set must reference only one table, the table must have a primary key; the SELECT must alias the original blob column name, specified as a string, to an alternate name; the SELECT must cover all columns that make up the primary key.	false	3.1.0
emulateUnsupportedPstmts	Should the driver detect prepared statements that are not supported by the server, and replace them with client-side emulated versions?	true	3.1.7
functionsNeverReturnBlobs	Should the driver always treat data from functions returning BLOBs as Strings - specifically to work around dubious metadata returned by the server for GROUP BY clauses?	false	5.0.8
generateSimpleParameterMetadata	Should the driver generate simplified parameter metadata for PreparedStatements when no metadata is available either because the server couldn't support preparing the statement, or server-side prepared statements are disabled?	false	5.0.5
ignoreNonTxTables	Ignore non-transactional table warning for rollback? (defaults to 'false').	false	3.0.9
jdbcCompliantTruncation	Should the driver throw java.sql.DataTruncation exceptions when data is truncated as is required by the JDBC specification when connected to a server that supports warnings (MySQL 4.1.0 and newer)? This property has no effect if the server sql-mode includes STRICT_TRANS_TABLES.	true	3.1.2
maxRows	The maximum number of rows to return (0, the default means return all rows).	-1	all versions

netTimeoutForStreamingResults	What value should the driver automatically set the server setting 'net_write_timeout' to when the streaming result sets feature is in use? (value has unit of seconds, the value '0' means the driver will not try and adjust this value)	600	5.1.0
noAccessToProcedureBodies	When determining procedure parameter types for CallableStatements, and the connected user can't access procedure bodies through "SHOW CREATE PROCEDURE" or select on mysql.proc should the driver instead create basic metadata (all parameters reported as IN VARCHARs, but allowing registerOutParameter() to be called on them anyway) instead of throwing an exception?	false	5.0.3
noDatetimeStringSync	Don't ensure that ResultSet.getString().equals(ResultSet.getString())	false	3.1.7
noTimezoneConversionForTimeType	Don't convert TIME values using the server timezone if 'useTimezone'='true'	false	5.0.0
nullCatalogMeansCurrent	When DatabaseMetadataMethods ask for a 'catalog' parameter, does the value null mean use the current catalog? (this is not JDBC-compliant, but follows legacy behavior from earlier versions of the driver)	true	3.1.8
nullNamePatternMatchesAll	Should DatabaseMetaMethods that accept *pattern parameters treat null the same as '%' (this is not JDBC-compliant, however older versions of the driver accepted this departure from the specification)	true	3.1.8
overrideSupportsIntegrityEnhancementFacility	Should the driver return "true" for DatabaseMetaMethods.supportsIntegrityEnhancementFacility() even if the database doesn't support it to workaround applications that require this method to return "true" to signal support of foreign keys, even though the SQL specification states that this facility contains much more than just foreign key support (one such application being OpenOffice)?	false	3.1.12
padCharsWithSpace	If a result set column has the CHAR type and the value does not fill the amount of characters specified in the DDL for the column, should the driver pad the remaining characters with space (for ANSI compliance)?	false	5.0.6
pedantic	Follow the JDBC spec to the letter.	false	3.0.0
pinGlobalTxToPhysicalConnection	When using XAConnections, should the driver ensure that operations on a given XID are always routed to the same physical connection? This allows the XAConnection to support "XA START ... JOIN" after "XA END" has been called	false	5.0.1
populateInsertRowWithDefaultValues	When using ResultSets that are CONCUR_UPDATABLE, should the driver pre-populate the "insert" row with default values from the DDL for the table used in the query so those values are immediately available for ResultSet accessors? This functionality requires a call to the database for metadata each time a result set of this type is created. If disabled (the default), the default values will be populated by the an internal call to refreshRow() which pulls back default values and/or values changed by triggers.	false	5.0.5
processEscapeCodesForPrepStmts	Should the driver process escape codes in queries that are prepared?	true	3.1.12
relaxAutoCommit	If the version of MySQL the driver connects to does not support transactions, still allow calls to commit(), rollback() and setAutoCommit() (true/false, defaults to 'false')?	false	2.0.13
retainStatementAfterResultSetClose	Should the driver retain the Statement reference in a ResultSet after ResultSet.close() has been called. This is not JDBC-compliant after JDBC-4.0.	false	3.1.11
rollbackOnPooledClose	Should the driver issue a rollback() when the logical connection in a pool is closed?	true	3.0.15
runningCTS13	Enables workarounds for bugs in Sun's JDBC compliance test suite	false	3.1.7

	version 1.3		
serverTimezone	Override detection/mapping of timezone. Used when timezone from server doesn't map to Java timezone		3.0.2
statementInterceptors	A comma-delimited list of classes that implement "" that should be placed "in between" query execution to influence the results. StatementInterceptors are "chainable", the results returned by the "current" interceptor will be passed on to the next in the chain, from left-to-right order, as specified in this property.		5.1.1
strictFloatingPoint	Used only in older versions of compliance test	false	3.0.0
strictUpdates	Should the driver do strict checking (all primary keys selected) of updatable result sets (true, false, defaults to 'true')?	true	3.0.4
tinyInt1isBit	Should the driver treat the datatype TINYINT(1) as the BIT type (because the server silently converts BIT -> TINYINT(1) when creating tables)?	true	3.0.16
transformedBitsBoolean	If the driver converts TINYINT(1) to a different type, should it use BOOLEAN instead of BIT for future compatibility with MySQL-5.0, as MySQL-5.0 has a BIT type?	false	3.1.9
treatUtilDateAsTimestamp	Should the driver treat java.util.Date as a TIMESTAMP for the purposes of PreparedStatement.setObject()?	true	5.0.5
ultraDevHack	Create PreparedStatements for prepareCall() when required, because UltraDev is broken and issues a prepareCall() for _all_ statements? (true/false, defaults to 'false')	false	2.0.3
useGmtMillisForDatetimes	Convert between session timezone and GMT before creating Date and Timestamp instances (value of "false" is legacy behavior, "true" leads to more JDBC-compliant behavior.	false	3.1.12
useHostsInPrivileges	Add '@hostname' to users in Database-MetaData.getColumn/TablePrivileges() (true/false), defaults to 'true'.	true	3.0.2
useInformationSchema	When connected to MySQL-5.0.7 or newer, should the driver use the INFORMATION_SCHEMA to derive information used by DatabaseMetaData?	false	5.0.0
useJDBCCompliantTimezoneShift	Should the driver use JDBC-compliant rules when converting TIME/TIMESTAMP/DATETIME values' timezone information for those JDBC arguments which take a java.util.Calendar argument? (Notice that this option is exclusive of the "use-Timezone=true" configuration option.)	false	5.0.0
useOldAliasMetadataBehavior	Should the driver use the legacy behavior for "AS" clauses on columns and tables, and only return aliases (if any) for ResultSet-MetaData.getColumnNames() or ResultSet-MetaData.getTableName() rather than the original column/table name?	false	5.0.4
useOldUTF8Behavior	Use the UTF-8 behavior the driver did when communicating with 4.0 and older servers	false	3.1.6
useOnlyServerErrorMessages	Don't prepend 'standard' SQLState error messages to error messages returned by the server.	true	3.0.15
useSSPSCompatibleTimezoneShift	If migrating from an environment that was using server-side prepared statements, and the configuration property "useJDBCCompliantTimezoneShift" set to "true", use compatible behavior when not using server-side prepared statements when sending TIMESTAMP values to the MySQL server.	false	5.0.5
useServerPrepStmts	Use server-side prepared statements if the server supports them?	false	3.1.0
useSqlStateCodes	Use SQL Standard state codes instead of 'legacy' X/Open/SQL state codes (true/false), default is 'true'	true	3.1.3
useStreamLengthsInPrepStmts	Honor stream length parameter in PreparedStatement/ResultSet.setXXXStream() method calls (true/false, defaults to 'true')?	true	3.0.2

useTimezone	Convert time/date types between client and server timezones (true/false, defaults to 'false')?	false	3.0.2
useUnbufferedInput	Don't use BufferedInputStream for reading data from the server	true	3.0.11
yearIsDateType	Should the JDBC driver treat the MySQL type "YEAR" as a java.sql.Date, or as a SHORT?	true	3.1.9
zeroDateTimeBehavior	What should happen when the driver encounters DATETIME values that are composed entirely of zeroes (used by MySQL to represent invalid dates)? Valid values are "exception", "round" and "convertToNull".	exception	3.1.4

Connector/J also supports access to MySQL via named pipes on Windows NT/2000/XP using the NamedPipeSocketFactory as a plugin-socket factory via the socketFactory property. If you don't use a namedPipePath property, the default of '\\.\pipe\MySQL' will be used. If you use the [NamedPipeSocketFactory](#), the hostname and port number values in the JDBC url will be ignored. You can enable this feature using:

```
socketFactory=com.mysql.jdbc.NamedPipeSocketFactory
```

Named pipes only work when connecting to a MySQL server on the same physical machine as the one the JDBC driver is being used on. In simple performance tests, it appears that named pipe access is between 30%-50% faster than the standard TCP/IP access.

You can create your own socket factories by following the example code in [com.mysql.jdbc.NamedPipeSocketFactory](#), or [com.mysql.jdbc.StandardSocketFactory](#).

26.4.4.2. JDBC API Implementation Notes

MySQL Connector/J passes all of the tests in the publicly-available version of Sun's JDBC compliance test suite. However, in many places the JDBC specification is vague about how certain functionality should be implemented, or the specification allows leeway in implementation.

This section gives details on a interface-by-interface level about how certain implementation decisions may affect how you use MySQL Connector/J.

- **Blob**

Starting with Connector/J version 3.1.0, you can emulate Blobs with locators by adding the property 'emulateLocators=true' to your JDBC URL. Using this method, the driver will delay loading the actual Blob data until you retrieve the other data and then use retrieval methods ([getInputStream\(\)](#), [getBytes\(\)](#), and so forth) on the blob data stream.

For this to work, you must use a column alias with the value of the column to the actual name of the Blob, for example:

```
SELECT id, 'data' as blob_data from blobtable
```

For this to work, you must also follow these rules:

- The **SELECT** must also reference only one table, the table must have a primary key.
- The **SELECT** must alias the original blob column name, specified as a string, to an alternate name.
- The **SELECT** must cover all columns that make up the primary key.

The Blob implementation does not allow in-place modification (they are copies, as reported by the [DatabaseMetaData.locatorsUpdateCopies\(\)](#) method). Because of this, you should use the corresponding [PreparedStatement.setBlob\(\)](#) or [ResultSet.updateBlob\(\)](#) (in the case of updatable result sets) methods to save changes back to the database.

MySQL Enterprise

MySQL Enterprise subscribers will find more information about type conversion in the Knowledge Base article, [Type Conversions Supported by MySQL Connector/J](#). To subscribe to MySQL Enterprise see <http://www.mysql.com/products/enterprise/advisors.html>.

- **CallableStatement**

Starting with Connector/J 3.1.1, stored procedures are supported when connecting to MySQL version 5.0 or newer via the `CallableStatement` interface. Currently, the `getParameterMetaData()` method of `CallableStatement` is not supported.

- **Clob**

The Clob implementation does not allow in-place modification (they are copies, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, you should use the `PreparedStatement.setClob()` method to save changes back to the database. The JDBC API does not have a `ResultSet.updateClob()` method.

- **Connection**

Unlike older versions of MM.MySQL the `isClosed()` method does not ping the server to determine if it is alive. In accordance with the JDBC specification, it only returns true if `closed()` has been called on the connection. If you need to determine if the connection is still valid, you should issue a simple query, such as `SELECT 1`. The driver will throw an exception if the connection is no longer valid.

- **DatabaseMetaData**

Foreign Key information (`getImportedKeys()/getExportedKeys()` and `getCrossReference()`) is only available from InnoDB tables. However, the driver uses `SHOW CREATE TABLE` to retrieve this information, so when other storage engines support foreign keys, the driver will transparently support them as well.

- **PreparedStatement**

PreparedStatements are implemented by the driver, as MySQL does not have a prepared statement feature. Because of this, the driver does not implement `getParameterMetaData()` or `getMetaData()` as it would require the driver to have a complete SQL parser in the client.

Starting with version 3.1.0 MySQL Connector/J, server-side prepared statements and binary-encoded result sets are used when the server supports them.

Take care when using a server-side prepared statement with **large** parameters that are set via `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()`, or `setClob()`. If you want to re-execute the statement with any large parameter changed to a non-large parameter, it is necessary to call `clearParameters()` and set all parameters again. The reason for this is as follows:

- During both server-side prepared statements and client-side emulation, large data is exchanged only when `PreparedStatement.execute()` is called.
- Once that has been done, the stream used to read the data on the client side is closed (as per the JDBC spec), and can't be read from again.
- If a parameter changes from large to non-large, the driver must reset the server-side state of the prepared statement to allow the parameter that is being changed to take the place of the prior large value. This removes all of the large data that has already been sent to the server, thus requiring the data to be re-sent, via the `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()` or `setClob()` methods.

Consequently, if you want to change the type of a parameter to a non-large one, you must call `clearParameters()` and set all parameters of the prepared statement again before it can be re-executed.

- **ResultSet**

By default, ResultSets are completely retrieved and stored in memory. In most cases this is the most efficient way to operate, and due to the design of the MySQL network protocol is easier to implement. If you are working with ResultSets that have a large number of rows or large values, and can not allocate heap space in your JVM for the memory required, you can tell the driver to stream the results back one row at a time.

To enable this functionality, you need to create a Statement instance in the following manner:

```
stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,  
                             java.sql.ResultSet.CONCUR_READ_ONLY);  
stmt.setFetchSize(Integer.MIN_VALUE);
```

The combination of a forward-only, read-only result set, with a fetch size of `Integer.MIN_VALUE` serves as a signal to the driver to stream result sets row-by-row. After this any result sets created with the statement will be retrieved row-by-row.

There are some caveats with this approach. You will have to read all of the rows in the result set (or close it) before you can issue any other queries on the connection, or an exception will be thrown.

The earliest the locks these statements hold can be released (whether they be `MyISAM` table-level locks or row-level locks in some other storage engine such as `InnoDB`) is when the statement completes.

If the statement is within scope of a transaction, then locks are released when the transaction completes (which implies that the statement needs to complete first). As with most other databases, statements are not complete until all the results pending on the statement are read or the active result set for the statement is closed.

Therefore, if using streaming results, you should process them as quickly as possible if you want to maintain concurrent access to the tables referenced by the statement producing the result set.

- **ResultSetMetaData**

The `isAutoIncrement()` method only works when using MySQL servers 4.0 and newer.

- **Statement**

When using versions of the JDBC driver earlier than 3.2.1, and connected to server versions earlier than 5.0.3, the `setFetchSize()` method has no effect, other than to toggle result set streaming as described above.

Connector/J 5.0.0 and later include support for both `Statement.cancel()` and `Statement.setQueryTimeout()`. Both require MySQL 5.0.0 or newer server, and require a separate connection to issue the `KILL QUERY` statement. In the case of `setQueryTimeout()`, the implementation creates an additional thread to handle the timeout functionality.

Note

Failures to cancel the statement for `setQueryTimeout()` may manifest themselves as `RuntimeException` rather than failing silently, as there is currently no way to unblock the thread that is executing the query being cancelled due to timeout expiration and have it throw the exception instead.

MySQL does not support SQL cursors, and the JDBC driver doesn't emulate them, so "setCursorName()" has no effect.

Connector/J 5.1.3 and later include two additional methods:

- `setLocalInfileInputStream()` sets an `InputStream` instance that will be used to send data to the MySQL server for a `LOAD DATA LOCAL INFILE` statement rather than a `FileInputStream` or `URLInputStream` that represents the path given as an argument to the statement.

This stream will be read to completion upon execution of a `LOAD DATA LOCAL INFILE` statement, and will automatically be closed by the driver, so it needs to be reset before each call to `execute*()` that would cause the MySQL server to request data to fulfill the request for `LOAD DATA LOCAL INFILE`.

If this value is set to `NULL`, the driver will revert to using a `FileInputStream` or `URLInputStream` as required.

- `getLocalInfileInputStream()` returns the `InputStream` instance that will be used to send data in response to a `LOAD DATA LOCAL INFILE` statement.

This method returns `NULL` if no such stream has been set via `setLocalInfileInputStream()`.

26.4.4.3. Java, JDBC and MySQL Types

MySQL Connector/J is flexible in the way it handles conversions between MySQL data types and Java data types.

In general, any MySQL data type can be converted to a `java.lang.String`, and any numerical type can be converted to any of the Java numerical types, although round-off, overflow, or loss of precision may occur.

Starting with Connector/J 3.1.0, the JDBC driver will issue warnings or throw `DataTruncation` exceptions as is required by the JDBC specification unless the connection was configured not to do so by using the property `jdbcCompliantTruncation` and setting it to `false`.

The conversions that are always guaranteed to work are listed in the following table:

Connection Properties - Miscellaneous.

These MySQL Data Types	Can always be converted to these Java types
CHAR, VARCHAR, BLOB, TEXT, ENUM, and SET	<code>java.lang.String</code> , <code>java.io.InputStream</code> , <code>java.io.Reader</code> , <code>java.sql.Blob</code> , <code>java.sql.Clob</code>
FLOAT, REAL, DOUBLE PRECISION, NUMERIC, DECIMAL, TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT	<code>java.lang.String</code> , <code>java.lang.Short</code> , <code>java.lang.Integer</code> , <code>java.lang.Long</code> , <code>java.lang.Double</code> , <code>java.math.BigDecimal</code>
DATE, TIME, DATETIME, TIMESTAMP	<code>java.lang.String</code> , <code>java.sql.Date</code> , <code>java.sql.Timestamp</code>

Note

Round-off, overflow or loss of precision may occur if you choose a Java numeric data type that has less precision or capacity than the MySQL data type you are converting to/from.

The `ResultSet.getObject()` method uses the type conversions between MySQL and Java types, following the JDBC specification where appropriate. The value returned by `ResultSetMetaData.getColumnClassName()` is also shown below. For more information on the `java.sql.Types` classes see [Java 2 Platform Types](#).

MySQL Types to Java Types for `ResultSet.getObject()`.

MySQL Type Name	Return value of <code>getColumnClassName</code>	Returned as Java Class
BIT(1) (new in MySQL-5.0)	BIT	<code>java.lang.Boolean</code>
BIT(> 1) (new in MySQL-5.0)	BIT	<code>byte[]</code>
TINYINT	TINYINT	<code>java.lang.Boolean</code> if the configuration property <code>tinyInt1isBit</code> is set to <code>true</code> (the default) and the storage size is 1, or <code>java.lang.Integer</code> if not.
BOOL, BOOLEAN	TINYINT	See TINYINT, above as these are aliases for TINYINT(1), currently.
SMALLINT[(M)] [UNSIGNED]	SMALLINT [UNSIGNED]	<code>java.lang.Integer</code> (regardless if UNSIGNED or not)
MEDIUMINT[(M)] [UNSIGNED]	MEDIUMINT [UNSIGNED]	<code>java.lang.Integer</code> , if UNSIGNED <code>java.lang.Long</code>
INT, INTEGER[(M)] [UNSIGNED]	INTEGER [UNSIGNED]	<code>java.lang.Integer</code> , if UNSIGNED <code>java.lang.Long</code>
BIGINT[(M)] [UNSIGNED]	BIGINT [UNSIGNED]	<code>java.lang.Long</code> , if UNSIGNED <code>java.math.BigInteger</code>
FLOAT[(M,D)]	FLOAT	<code>java.lang.Float</code>
DOUBLE[(M,B)]	DOUBLE	<code>java.lang.Double</code>
DECIMAL[(M[,D))]	DECIMAL	<code>java.math.BigDecimal</code>
DATE	DATE	<code>java.sql.Date</code>
DATETIME	DATETIME	<code>java.sql.Timestamp</code>
TIMESTAMP[(M)]	TIMESTAMP	<code>java.sql.Timestamp</code>
TIME	TIME	<code>java.sql.Time</code>
YEAR[(2 4)]	YEAR	If <code>yearIsDateType</code> configuration property is set to false, then the returned object type is <code>java.sql.Short</code> . If set to true (the default) then an object of type <code>java.sql.Date</code> (with the date set to January 1st, at midnight).
CHAR(M)	CHAR	<code>java.lang.String</code> (unless the character set for the column is BINARY, then <code>byte[]</code> is returned).

VARCHAR(M) [BINARY]	VARCHAR	java.lang.String (unless the character set for the column is BINARY, then byte[] is returned).
BINARY(M)	BINARY	byte[]
VARBINARY(M)	VARBINARY	byte[]
TINYBLOB	TINYBLOB	byte[]
TINYTEXT	VARCHAR	java.lang.String
BLOB	BLOB	byte[]
TEXT	VARCHAR	java.lang.String
MEDIUMBLOB	MEDIUMBLOB	byte[]
MEDIUMTEXT	VARCHAR	java.lang.String
LOBLOB	LOBLOB	byte[]
LONGTEXT	VARCHAR	java.lang.String
ENUM('value1','value2',...)	CHAR	java.lang.String
SET('value1','value2',...)	CHAR	java.lang.String

26.4.4.4. Using Character Sets and Unicode

All strings sent from the JDBC driver to the server are converted automatically from native Java Unicode form to the client character encoding, including all queries sent via [Statement.execute\(\)](#), [Statement.executeUpdate\(\)](#), [Statement.executeQuery\(\)](#) as well as all [PreparedStatement](#) and [CallableStatement](#) parameters with the exclusion of parameters set using [setBytes\(\)](#), [setBinaryStream\(\)](#), [setAsciiStream\(\)](#), [setUnicodeStream\(\)](#) and [setBlob\(\)](#).

Prior to MySQL Server 4.1, Connector/J supported a single character encoding per connection, which could either be automatically detected from the server configuration, or could be configured by the user through the [useUnicode](#) and [characterEncoding](#) properties.

Starting with MySQL Server 4.1, Connector/J supports a single character encoding between client and server, and any number of character encodings for data returned by the server to the client in [ResultSets](#).

The character encoding between client and server is automatically detected upon connection. The encoding used by the driver is specified on the server via the [character_set](#) system variable for server versions older than 4.1.0 and [character_set_server](#) for server versions 4.1.0 and newer. For more information, see [Section 9.3.1, “Server Character Set and Collation”](#).

To override the automatically-detected encoding on the client side, use the [characterEncoding](#) property in the URL used to connect to the server.

When specifying character encodings on the client side, Java-style names should be used. The following table lists Java-style names for MySQL character sets:

MySQL to Java Encoding Name Translations.

MySQL Character Set Name	Java-Style Character Encoding Name
ascii	US-ASCII
big5	Big5
gbk	GBK
sjis	SJIS (or Cp932 or MS932 for MySQL Server < 4.1.11)
cp932	Cp932 or MS932 (MySQL Server > 4.1.11)
gb2312	EUC_CN
ujis	EUC_JP
euckr	EUC_KR
latin1	ISO8859_1
latin2	ISO8859_2
greek	ISO8859_7

hebrew	ISO8859_8
cp866	Cp866
tis620	TIS620
cp1250	Cp1250
cp1251	Cp1251
cp1257	Cp1257
macroman	MacRoman
macce	MacCentralEurope
utf8	UTF-8
ucs2	UnicodeBig

Warning

Do not issue the query 'set names' with Connector/J, as the driver will not detect that the character set has changed, and will continue to use the character set detected during the initial connection setup.

To allow multiple character sets to be sent from the client, the UTF-8 encoding should be used, either by configuring `utf8` as the default server character set, or by configuring the JDBC driver to use UTF-8 through the `characterEncoding` property.

26.4.4.5. Connecting Securely Using SSL

SSL in MySQL Connector/J encrypts all data (other than the initial handshake) between the JDBC driver and the server. The performance penalty for enabling SSL is an increase in query processing time between 35% and 50%, depending on the size of the query, and the amount of data it returns.

For SSL Support to work, you must have the following:

- A JDK that includes JSSE (Java Secure Sockets Extension), like JDK-1.4.1 or newer. SSL does not currently work with a JDK that you can add JSSE to, like JDK-1.2.x or JDK-1.3.x due to the following JSSE bug: <http://developer.java.sun.com/developer/bugParade/bugs/4273544.html>
- A MySQL server that supports SSL and has been compiled and configured to do so, which is MySQL-4.0.4 or later, see [Section 5.8.7, “Using Secure Connections”](#), for more information.
- A client certificate (covered later in this section)

You will first need to import the MySQL server CA Certificate into a Java truststore. A sample MySQL server CA Certificate is located in the `SSL` subdirectory of the MySQL source distribution. This is what SSL will use to determine if you are communicating with a secure MySQL server.

To use Java's `keytool` to create a truststore in the current directory, and import the server's CA certificate (`cacert.pem`), you can do the following (assuming that `keytool` is in your path. The `keytool` should be located in the `bin` subdirectory of your JDK or JRE):

```
shell> keytool -import -alias mysqlServerCACert \
               -file cacert.pem -keystore truststore
```

Keytool will respond with the following information:

```
Enter keystore password: *****
Owner: EMAILADDRESS=walrus@example.com, CN=walrus,
      O=MySQL AB, L=Orenburg, ST=Some-State, C=RU
Issuer: EMAILADDRESS=walrus@example.com, CN=walrus,
        O=MySQL AB, L=Orenburg, ST=Some-State, C=RU
Serial number: 0
Valid from:
  Fri Aug 02 16:55:53 CDT 2002 until: Sat Aug 02 16:55:53 CDT 2003
Certificate fingerprints:
  MD5:  61:91:A0:F2:03:07:61:7A:81:38:66:DA:19:C4:8D:AB
  SHA1: 25:77:41:05:D5:AD:99:8C:14:8C:CA:68:9C:2F:B8:89:C3:34:4D:6C
Trust this certificate? [no]: yes
```

Certificate was added to keystore

You will then need to generate a client certificate, so that the MySQL server knows that it is talking to a secure client:

```
shell> keytool -genkey -keyalg rsa \
  -alias mysqlClientCertificate -keystore keystore
```

Keytool will prompt you for the following information, and create a keystore named **keystore** in the current directory.

You should respond with information that is appropriate for your situation:

```
Enter keystore password: *****
What is your first and last name?
[Unknown]: Matthews
What is the name of your organizational unit?
[Unknown]: Software Development
What is the name of your organization?
[Unknown]: MySQL AB
What is the name of your City or Locality?
[Unknown]: Flossmoor
What is the name of your State or Province?
[Unknown]: IL
What is the two-letter country code for this unit?
[Unknown]: US
Is <CN=Matthews, OU=Software Development, O=MySQL AB,
L=Flossmoor, ST=IL, C=US> correct?
[no]: y
Enter key password for <mysqlClientCertificate>
(RETURN if same as keystore password):
```

Finally, to get JSSE to use the keystore and truststore that you have generated, you need to set the following system properties when you start your JVM, replacing `path_to_keystore_file` with the full path to the keystore file you created, `path_to_truststore_file` with the path to the truststore file you created, and using the appropriate password values for each property. You can do this either on the command line:

```
-Djavax.net.ssl.keyStore=path_to_keystore_file
-Djavax.net.ssl.keyStorePassword=password
-Djavax.net.ssl.trustStore=path_to_truststore_file
-Djavax.net.ssl.trustStorePassword=password
```

Or you can set the values directly within the application:

```
System.setProperty("javax.net.ssl.keyStore", "path_to_keystore_file");
System.setProperty("javax.net.ssl.keyStorePassword", "password");
System.setProperty("javax.net.ssl.trustStore", "path_to_truststore_file");
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

You will also need to set `useSSL` to **true** in your connection parameters for MySQL Connector/J, either by adding `useSSL=true` to your URL, or by setting the property `useSSL` to **true** in the `java.util.Properties` instance you pass to `DriverManager.getConnection()`.

You can test that SSL is working by turning on JSSE debugging (as detailed below), and look for the following key events:

```
*** ClientHello, v3.1
RandomCookie: GMT: 1018531834 bytes = { 199, 148, 180, 215, 74, 12, »
54, 244, 0, 168, 55, 103, 215, 64, 16, 138, 225, 190, 132, 153, 2, »
217, 219, 239, 202, 19, 121, 78 }
Session ID: {}
Cipher Suites: { 0, 5, 0, 4, 0, 9, 0, 10, 0, 18, 0, 19, 0, 3, 0, 17 }
Compression Methods: { 0 }
***
[write] MD5 and SHA1 hashes: len = 59
0000: 01 00 00 37 03 01 3D B6 90 FA C7 94 B4 D7 4A 0C ...7..=.....J.
0010: 36 F4 00 A8 37 67 D7 40 10 8A E1 BE 84 99 02 D9 6...7g.@.....
0020: DB EF CA 13 79 4E 00 00 10 00 05 00 04 00 09 00 ....yN.....
0030: 0A 00 12 00 13 00 03 00 11 01 00 .....
main, WRITE: SSL v3.1 Handshake, length = 59
main, READ: SSL v3.1 Handshake, length = 74
*** ServerHello, v3.1
RandomCookie: GMT: 1018577560 bytes = { 116, 50, 4, 103, 25, 100, 58, »
202, 79, 185, 178, 100, 215, 66, 254, 21, 83, 187, 190, 42, 170, 3, »
132, 110, 82, 148, 160, 92 }
Session ID: {163, 227, 84, 53, 81, 127, 252, 254, 178, 179, 68, 63, »
182, 158, 30, 11, 150, 79, 170, 76, 255, 92, 15, 226, 24, 17, 177, »
```



```

    219, 158, 177, 187, 143}
Cipher Suite: { 0, 5 }
Compression Method: 0
***
%% Created: [Session-1, SSL_RSA_WITH_RC4_128_SHA]
** SSL_RSA_WITH_RC4_128_SHA
[read] MD5 and SHA1 hashes: len = 74
0000: 02 00 00 46 03 01 3D B6 43 98 74 32 04 67 19 64 ...F...C.t2.g.d
0010: 3A CA 4F B9 B2 64 D7 42 FE 15 53 BB BE 2A AA 03 ..0..d.B..S..*..
0020: 84 6E 52 94 A0 5C 20 A3 E3 54 35 51 7F FC FE B2 .nR..\..T5Q....
0030: B3 44 3F B6 9E 1E 0B 96 4F AA 4C FF 5C 0F E2 18 .D?.....0.L.\...
0040: 11 B1 DB 9E B1 BB 8F 00 05 00 .....
main, READ: SSL v3.1 Handshake, length = 1712
...

```

JSSE provides debugging (to STDOUT) when you set the following system property: `-Djavax.net.debug=all`. This will tell you what keystores and truststores are being used, as well as what is going on during the SSL handshake and certificate exchange. It will be helpful when trying to determine what is not working when trying to get an SSL connection to happen.

26.4.4.6. Using Master/Slave Replication with ReplicationConnection

Starting with Connector/J 3.1.7, we've made available a variant of the driver that will automatically send queries to a read/write master, or a failover or round-robin loadbalanced set of slaves based on the state of `Connection.getReadOnly()`.

An application signals that it wants a transaction to be read-only by calling `Connection.setReadOnly(true)`, this replication-aware connection will use one of the slave connections, which are load-balanced per-vm using a round-robin scheme (a given connection is sticky to a slave unless that slave is removed from service). If you have a write transaction, or if you have a read that is time-sensitive (remember, replication in MySQL is asynchronous), set the connection to be not read-only, by calling `Connection.setReadOnly(false)` and the driver will ensure that further calls are sent to the master MySQL server. The driver takes care of propagating the current state of autocommit, isolation level, and catalog between all of the connections that it uses to accomplish this load balancing functionality.

To enable this functionality, use the "`com.mysql.jdbc.ReplicationDriver`" class when configuring your application server's connection pool or when creating an instance of a JDBC driver for your standalone application. Because it accepts the same URL format as the standard MySQL JDBC driver, `ReplicationDriver` does not currently work with `java.sql.DriverManager`-based connection creation unless it is the only MySQL JDBC driver registered with the `DriverManager`.

Here is a short, simple example of how `ReplicationDriver` might be used in a standalone application.

```

import java.sql.Connection;
import java.sql.ResultSet;
import java.util.Properties;

import com.mysql.jdbc.ReplicationDriver;

public class ReplicationDriverDemo {

    public static void main(String[] args) throws Exception {
        ReplicationDriver driver = new ReplicationDriver();

        Properties props = new Properties();

        // We want this for failover on the slaves
        props.put("autoReconnect", "true");

        // We want to load balance between the slaves
        props.put("roundRobinLoadBalance", "true");

        props.put("user", "foo");
        props.put("password", "bar");

        //
        // Looks like a normal MySQL JDBC url, with a
        // comma-separated list of hosts, the first
        // being the 'master', the rest being any number
        // of slaves that the driver will load balance against
        //
        Connection conn =
            driver.connect("jdbc:mysql://master,slave1,slave2,slave3/test",
                props);

        //
        // Perform read/write work on the master
        // by setting the read-only flag to "false"
        //
        conn.setReadOnly(false);
        conn.setAutoCommit(false);
    }
}

```

```

conn.createStatement().executeUpdate("UPDATE some_table ...");
conn.commit();

//
// Now, do a query from a slave, the driver automatically picks one
// from the list
//

conn.setReadOnly(true);

ResultSet rs =
    conn.createStatement().executeQuery("SELECT a,b FROM alt_table");
    .....
}
}

```

26.4.4.7. Mapping MySQL Error Numbers to SQLStates

The table below provides a mapping of the MySQL Error Numbers to [SQL States](#)

Table 26.1. Mapping of MySQL Error Numbers to SQLStates

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1022	ER_DUP_KEY	S1000	23000
1037	ER_OUT_OFMEMORY	S1001	HY001
1038	ER_OUT_OF_SORT_MEMORY	S1001	HY001
1040	ER_CON_COUNT_ERROR	08004	08004
1042	ER_BAD_HOST_ERROR	08004	08S01
1043	ER_HANDSHAKE_ERROR	08004	08S01
1044	ER_DBACCESS_DENIED_ERROR	S1000	42000
1045	ER_ACCESS_DENIED_ERROR	28000	28000
1047	ER_UNKNOWN_COM_ERROR	08S01	HY000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQL State	SQL Standard SQL State
1050	ER_TABLE_EXISTS_ERROR	S1000	42S01
1051	ER_BAD_TABLE_ERROR	42S02	42S02
1052	ER_NON_UNIQ_ERROR	S1000	23000
1053	ER_SERVER_SHUTDOWN	S1000	08S01
1054	ER_BAD_FIELD_ERROR	S0022	42S22
1055	ER_WRONG_FIELD_WITH_GROUP	S1009	42000
1056	ER_WRONG_GROUP_FIELD	S1009	42000
1057	ER_WRONG_SUM_SELECT	S1009	42000
1058	ER_WRONG_VALUE_COUNT	21S01	21S01
1059	ER_TOO_LONG_IDENT	S1009	42000
1060	ER_DUP_FIELD_NAME	S1009	42S21
1061	ER_DUP_KEYNAME	S1009	42000
1062	ER_DUP_ENTRY	S1009	23000
1063	ER_WRONG_FIELD_SPEC	S1009	42000
1064	ER_PARSE_ERROR	42000	42000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQL State	SQL Standard SQL State
	OR		
1065	ER_EMPTYPE_QUERY	42000	42000
1066	ER_NONUNIQUE_TABLE	S1009	42000
1067	ER_INVALID_ID_DEFAULT	S1009	42000
1068	ER_MULTIPLE_PRI_KEY	S1009	42000
1069	ER_TOO_MANY_KEYS	S1009	42000
1070	ER_TOO_MANY_KEY_PARTS	S1009	42000
1071	ER_TOO_LONG_KEY	S1009	42000
1072	ER_KEY_COLUMN_DOES_NOT_EXITS	S1009	42000
1073	ER_BLOB_USED_AS_KEY	S1009	42000
1074	ER_TOO_BIG_FIELDLENGTH	S1009	42000
1075	ER_WRONG_AUTO_KEY	S1009	42000
1080	ER_FORGING_CLOSE	S1000	08S01
1081	ER_IPSOCK_ERROR	08S01	08S01
1082	ER_NO_SUCH_INDEX	S1009	42S12

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQL State	SQL Standard SQL State
1083	ER_WRONG_FIELD_TERMINATORS	S1009	42000
1084	ER_BLOCK_AND_NO_TERMINATED	S1009	42000
1090	ER_CANT_REMOVE_ALL_FIELDS	S1000	42000
1091	ER_CANT_DROP_FIELD_OR_KEY	S1000	42000
1101	ER_BLOCK_CANT_HAVE_DEFAULT	S1000	42000
1102	ER_WRONG_DB_NAME	S1000	42000
1103	ER_WRONG_TABLE_NAME	S1000	42000
1104	ER_TOO_BIG_SELECT	S1000	42000
1106	ER_UNKNOWN_PROCEDURE	S1000	42000
1107	ER_WRONG_PARAM_COUNT_TO_PROCEDURE	S1000	42000
1109	ER_UNKNOWN_TABLE	S1000	42S02
1110	ER_FIELD_SPECIFIED	S1000	42000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
	FIED_TWICE		
1112	ER_UNSPUPPORTED_EXTENSION	S1000	42000
1113	ER_TABLE_MUST_HAVE_COLUMNS	S1000	42000
1115	ER_UNKNOWN_CHARACTER_SET	S1000	42000
1118	ER_TOO_BIG_ROWSIZE	S1000	42000
1120	ER_WRONG OUTER JOIN	S1000	42000
1121	ER_NULL_COLUMN_IN_INDEX	S1000	42000
1129	ER_HOST_IS_BLOCKED	08004	HY000
1130	ER_HOST_NOT_PRIVILEGED	08004	HY000
1131	ER_PASSWORD_ANONYMOUS_USER	S1000	42000
1132	ER_PASSWORD_NOT_ALLOWED	S1000	42000
1133	ER_PASSWORD_NO_MATCH	S1000	42000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQL State	SQL Standard SQL State
1136	ER_WRONG_VALUE_COUNT_ON_ROW	S1000	21S01
1138	ER_INVALID_ID_USE_OF_NULL	S1000	42000
1139	ER_REGEXP_ERROR	S1000	42000
1140	ER_MIX_OF_GROUP_FUNC_AND_FIELDS	S1000	42000
1141	ER_NONEXISTING_GRANT	S1000	42000
1142	ER_TABLEACCESS_DENIED_ERROR	S1000	42000
1143	ER_COLUMNACCESS_DENIED_ERROR	S1000	42000
1144	ER_ILLEGAL_GRANT_FOR_TABLE	S1000	42000
1145	ER_GRANT_WRONG_HOST_OR_USER	S1000	42000
1146	ER_NO_SUCH_TABLE	S1000	42S02
1147	ER_NONEXISTING_TABLE_GRANT	S1000	42000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQL State	SQL Standard SQL State
	ANT		
1148	ER_NOT_ALLOWED_COMMAND	S1000	42000
1149	ER_SYNTAX_ERROR	S1000	42000
1152	ER_ABORTING_CONNECTION	S1000	08S01
1153	ER_NET_PACKET_TOO_LARGE	S1000	08S01
1154	ER_NET_READ_ERROR_FROM_PIPE	S1000	08S01
1155	ER_NET_FCNTL_ERROR	S1000	08S01
1156	ER_NET_PACKETS_OUT_OF_ORDER	S1000	08S01
1157	ER_NET_UNCOMPRESS_ERROR	S1000	08S01
1158	ER_NET_READ_ERROR	S1000	08S01
1159	ER_NET_READ_INTERRUPTED	S1000	08S01
1160	ER_NET_ERROR_ON_WRITE	S1000	08S01
1161	ER_NET_WRITE_INTERRUPTED	S1000	08S01

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQL State	SQL Standard SQL State
1162	ER_TOO_LONG_STRING	S1000	42000
1163	ER_TABLE_CANNOT_HANDLE_BLOCK	S1000	42000
1164	ER_TABLE_CANNOT_HANDLE_AUTO_INCREMENT	S1000	42000
1166	ER_WRONG_COLUMN_NAME	S1000	42000
1167	ER_WRONG_KEY_COLUMN	S1000	42000
1169	ER_DUP_UNIQUE	S1000	23000
1170	ER_BLOCK_KEY_WITHOUT_LENGTH	S1000	42000
1171	ER_PRIMARY_CANT_HAVE_NULL	S1000	42000
1172	ER_TOO_MANY_ROWS	S1000	42000
1173	ER_REQUIRES_PRIMARY_KEY	S1000	42000
1177	ER_CHECK_NO_SUCH_TABLE	S1000	42000
1178	ER_CHECK_NOT_IMPLEMENTED	S1000	42000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQL State	SQL Standard SQL State
1179	ER_CANT_DO_THIS_DURING_AN_TRANSACTION	S1000	25000
1184	ER_NEW_ABORTING_CONNECTION	S1000	08S01
1189	ER_MASTER_NOT_READ	S1000	08S01
1190	ER_MASTER_NOT_WRITE	S1000	08S01
1203	ER_TOO_MANY_USER_CONNECTIONS	S1000	42000
1205	ER_LOCK_WAIT_TIMEOUT	41000	41000
1207	ER_READ_ONLY_TRANSACTION	S1000	25000
1211	ER_NO_PERMISSION_TO_CREATE_USER	S1000	42000
1213	ER_LOCK_DEADLOCK	41000	40001
1216	ER_NO_REFERENCED_ROW	S1000	23000
1217	ER_ROW_IS_REFERED	S1000	23000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQL State	SQL Standard SQL State
	ENCED		
1218	ER_CONNECT_TO_MASTER	S1000	08S01
1222	ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT	S1000	21000
1226	ER_USER_LIMIT_REACHED	S1000	42000
1230	ER_NO_DEFAULT	S1000	42000
1231	ER_WRONG_VALUE_FOR_VAR	S1000	42000
1232	ER_WRONG_TYPE_FOR_VAR	S1000	42000
1234	ER_CANT_USE_OPTION_HERE	S1000	42000
1235	ER_NOT_SUPPORTED_YET	S1000	42000
1239	ER_WRONG_FK_DEF	S1000	42000
1241	ER_OPERAND_COLUMNS	S1000	21000
1242	ER_SUBQUERY_NO_1_ROW	S1000	21000
1247	ER_ILLEGAL_RE	S1000	42S22

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQL State	SQL Standard SQL State
	ER_FER- ENCE		
1248	ER_DERIVED_MUST_HAVE_ALIAS	S1000	42000
1249	ER_SELECT_REJECTED	S1000	01000
1250	ER_TABLE_NAME_NOT_ALLOWED_HERE	S1000	42000
1251	ER_NOT_SUPPORTED_AUTH_MODE	S1000	08004
1252	ER_SPATIAL_CANT_HAVE_NULL	S1000	42000
1253	ER_COLLATION_CHARSET_MISMATCH	S1000	42000
1261	ER_WARN_TOO_FEW_RECORDS	S1000	01000
1262	ER_WARN_TOO_MANY_RECORDS	S1000	01000
1263	ER_WARN_NULL_TO_NOTNULL	S1000	01000
1264	ER_WARN_DATA_OUT_OF_RANGE	S1000	01000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQL State	SQL Standard SQL State
1265	ER_WARN_DATA_TRUNCATED	S1000	01000
1280	ER_WRONG_NAME_FOR_INDEX	S1000	42000
1281	ER_WRONG_NAME_FOR_CATALOG	S1000	42000
1286	ER_UNKNOWN_STORAGE_ENGINE	S1000	42000

26.4.5. Connector/J Notes and Tips

26.4.5.1. Basic JDBC Concepts

This section provides some general JDBC background.

26.4.5.1.1. Connecting to MySQL Using the `DriverManager` Interface

When you are using JDBC outside of an application server, the `DriverManager` class manages the establishment of Connections.

The `DriverManager` needs to be told which JDBC drivers it should try to make Connections with. The easiest way to do this is to use `Class.forName()` on the class that implements the `java.sql.Driver` interface. With MySQL Connector/J, the name of this class is `com.mysql.jdbc.Driver`. With this method, you could use an external configuration file to supply the driver class name and driver parameters to use when connecting to a database.

The following section of Java code shows how you might register MySQL Connector/J from the `main()` method of your application:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

// Notice, do not import com.mysql.jdbc.*
// or you will have problems!

public class LoadDriver {
    public static void main(String[] args) {
        try {
            // The newInstance() call is a work around for some
            // broken Java implementations

            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception ex) {
            // handle the error
        }
    }
}
```

After the driver has been registered with the `DriverManager`, you can obtain a `Connection` instance that is connected to a particular database by calling `DriverManager.getConnection()`:

Example 26.1. Obtaining a connection from the `DriverManager`

This example shows how you can obtain a `Connection` instance from the `DriverManager`. There are a few different signatures for the `getConnection()` method. You should see the API documentation that comes with your JDK for more specific information on how to use them.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

...
try {
    Connection conn =
        DriverManager.getConnection("jdbc:mysql://localhost/test?" +
                                   "user=monty&password=greatsqlldb");

    // Do something with the Connection

} catch (SQLException ex) {
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
```

Once a `Connection` is established, it can be used to create `Statement` and `PreparedStatement` objects, as well as retrieve metadata about the database. This is explained in the following sections.

26.4.5.1.2. Using Statements to Execute SQL

`Statement` objects allow you to execute basic SQL queries and retrieve the results through the `ResultSet` class which is described later.

To create a `Statement` instance, you call the `createStatement()` method on the `Connection` object you have retrieved via one of the `DriverManager.getConnection()` or `DataSource.getConnection()` methods described earlier.

Once you have a `Statement` instance, you can execute a `SELECT` query by calling the `executeQuery(String)` method with the SQL you want to use.

To update data in the database, use the `executeUpdate(String SQL)` method. This method returns the number of rows affected by the update statement.

If you don't know ahead of time whether the SQL statement will be a `SELECT` or an `UPDATE/INSERT`, then you can use the `execute(String SQL)` method. This method will return true if the SQL query was a `SELECT`, or false if it was an `UPDATE`, `INSERT`, or `DELETE` statement. If the statement was a `SELECT` query, you can retrieve the results by calling the `getResultSet()` method. If the statement was an `UPDATE`, `INSERT`, or `DELETE` statement, you can retrieve the affected rows count by calling `getUpdateCount()` on the `Statement` instance.

Example 26.2. Using `java.sql.Statement` to execute a `SELECT` query

```
// assume that conn is an already created JDBC connection
Statement stmt = null;
ResultSet rs = null;

try {
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT foo FROM bar");

    // or alternatively, if you don't know ahead of time that
    // the query will be a SELECT...

    if (stmt.execute("SELECT foo FROM bar")) {
        rs = stmt.getResultSet();
    }

    // Now do something with the ResultSet ....
} finally {
    // it is a good idea to release
    // resources in a finally{} block
    // in reverse-order of their creation
    // if they are no-longer needed
```

```

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) { // ignore }
        rs = null;
    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) { // ignore }
        stmt = null;
    }
}

```

26.4.5.1.3. Using `CallableStatements` to Execute Stored Procedures

Starting with MySQL server version 5.0 when used with Connector/J 3.1.1 or newer, the `java.sql.CallableStatement` interface is fully implemented with the exception of the `getParameterMetaData()` method.

For more information on MySQL stored procedures, please refer to <http://dev.mysql.com/doc/mysql/en/stored-procedures.html>.

Connector/J exposes stored procedure functionality through JDBC's `CallableStatement` interface.

Note

Current versions of MySQL server do not return enough information for the JDBC driver to provide result set metadata for callable statements. This means that when using `CallableStatement`, `ResultSetMetaData` may return `NULL`.

The following example shows a stored procedure that returns the value of `inOutParam` incremented by 1, and the string passed in via `inputParam` as a `ResultSet`:

Example 26.3. Stored Procedures

```

CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), \
                        INOUT inOutParam INT)
BEGIN
    DECLARE z INT;
    SET z = inOutParam + 1;
    SET inOutParam = z;

    SELECT inputParam;

    SELECT CONCAT('zyxw', inputParam);
END

```

To use the `demoSp` procedure with Connector/J, follow these steps:

1. Prepare the callable statement by using `Connection.prepareCall()`.

Notice that you have to use JDBC escape syntax, and that the parentheses surrounding the parameter placeholders are not optional:

Example 26.4. Using `Connection.prepareCall()`

```

import java.sql.CallableStatement;

...

//
// Prepare a call to the stored procedure 'demoSp'
// with two parameters
//
// Notice the use of JDBC-escape syntax ({call ...})
//

CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}");

```

```
cStmt.setString(1, "abcdefg");
```

Note

`Connection.prepareCall()` is an expensive method, due to the metadata retrieval that the driver performs to support output parameters. For performance reasons, you should try to minimize unnecessary calls to `Connection.prepareCall()` by reusing `CallableStatement` instances in your code.

2. Register the output parameters (if any exist)

To retrieve the values of output parameters (parameters specified as `OUT` or `INOUT` when you created the stored procedure), JDBC requires that they be specified before statement execution using the various `registerOutputParameter()` methods in the `CallableStatement` interface:

Example 26.5. Registering output parameters

```
import java.sql.Types;
...
//
// Connector/J supports both named and indexed
// output parameters. You can register output
// parameters using either method, as well
// as retrieve output parameters using either
// method, regardless of what method was
// used to register them.
//
// The following examples show how to use
// the various methods of registering
// output parameters (you should of course
// use only one registration per parameter).
//
//
// Registers the second parameter as output, and
// uses the type 'INTEGER' for values returned from
// getObject()
//
cStmt.registerOutParameter(2, Types.INTEGER);
//
// Registers the named parameter 'inOutParam', and
// uses the type 'INTEGER' for values returned from
// getObject()
//
cStmt.registerOutParameter("inOutParam", Types.INTEGER);
...
```

3. Set the input parameters (if any exist)

Input and in/out parameters are set as for `PreparedStatement` objects. However, `CallableStatement` also supports setting parameters by name:

Example 26.6. Setting `CallableStatement` input parameters

```
...
//
// Set a parameter by index
//
cStmt.setString(1, "abcdefg");
//
// Alternatively, set a parameter using
// the parameter name
//
cStmt.setString("inputParameter", "abcdefg");
```



```

//
// Set the 'in/out' parameter using an index
//
cStmt.setInt(2, 1);

//
// Alternatively, set the 'in/out' parameter
// by name
//
cStmt.setInt("inOutParam", 1);

...

```

4. Execute the `CallableStatement`, and retrieve any result sets or output parameters.

Although `CallableStatement` supports calling any of the `Statement` execute methods (`executeUpdate()`, `executeQuery()` or `execute()`), the most flexible method to call is `execute()`, as you do not need to know ahead of time if the stored procedure returns result sets:

Example 26.7. Retrieving results and output parameter values

```

...

boolean hadResults = cStmt.execute();

//
// Process all returned result sets
//
while (hadResults) {
    ResultSet rs = cStmt.getResultSet();

    // process result set
    ...

    hadResults = rs.getMoreResults();
}

//
// Retrieve output parameters
//
// Connector/J supports both index-based and
// name-based retrieval
//

int outputValue = cStmt.getInt(2); // index-based
outputValue = cStmt.getInt("inOutParam"); // name-based

...

```

26.4.5.1.4. Retrieving `AUTO_INCREMENT` Column Values

Before version 3.0 of the JDBC API, there was no standard way of retrieving key values from databases that supported auto increment or identity columns. With older JDBC drivers for MySQL, you could always use a MySQL-specific method on the `Statement` interface, or issue the query `SELECT LAST_INSERT_ID()` after issuing an `INSERT` to a table that had an `AUTO_INCREMENT` key. Using the MySQL-specific method call isn't portable, and issuing a `SELECT` to get the `AUTO_INCREMENT` key's value requires another round-trip to the database, which isn't as efficient as possible. The following code snippets demonstrate the three different ways to retrieve `AUTO_INCREMENT` values. First, we demonstrate the use of the new JDBC-3.0 method `getGeneratedKeys()` which is now the preferred method to use if you need to retrieve `AUTO_INCREMENT` keys and have access to JDBC-3.0. The second example shows how you can retrieve the same value using a standard `SELECT LAST_INSERT_ID()` query. The final example shows how updatable result sets can retrieve the `AUTO_INCREMENT` value when using the `insertRow()` method.

Example 26.8. Retrieving `AUTO_INCREMENT` column values using `Statement.getGeneratedKeys()`

```

Statement stmt = null;
ResultSet rs = null;

```

```

try {
    //
    // Create a Statement instance that we can use for
    // 'normal' result sets assuming you have a
    // Connection 'conn' to a MySQL database already
    // available

    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                                java.sql.ResultSet.CONCUR_UPDATABLE);

    //
    // Issue the DDL queries for the table for this example
    //
    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
    //
    stmt.executeUpdate(
        "INSERT INTO autoIncTutorial (dataField) "
        + "values ('Can I Get the Auto Increment Field?')",
        Statement.RETURN_GENERATED_KEYS);

    //
    // Example of using Statement.getGeneratedKeys()
    // to retrieve the value of an auto-increment
    // value
    //
    int autoIncKeyFromApi = -1;
    rs = stmt.getGeneratedKeys();
    if (rs.next()) {
        autoIncKeyFromApi = rs.getInt(1);
    } else {
        // throw an exception from here
    }

    rs.close();
    rs = null;

    System.out.println("Key returned from getGeneratedKeys(): "
        + autoIncKeyFromApi);
} finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
}

```

Example 26.9. Retrieving **AUTO_INCREMENT** column values using **SELECT LAST_INSERT_ID()**

```

Statement stmt = null;
ResultSet rs = null;

try {
    //
    // Create a Statement instance that we can use for
    // 'normal' result sets.

```

```

    stmt = conn.createStatement();

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ( "
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
    //

    stmt.executeUpdate(
        "INSERT INTO autoIncTutorial (dataField) "
        + "values ('Can I Get the Auto Increment Field?')");

    //
    // Use the MySQL LAST_INSERT_ID()
    // function to do the same thing as getGeneratedKeys()
    //

    int autoIncKeyFromFunc = -1;
    rs = stmt.executeQuery("SELECT LAST_INSERT_ID()");

    if (rs.next()) {
        autoIncKeyFromFunc = rs.getInt(1);
    } else {
        // throw an exception from here
    }

    rs.close();

    System.out.println("Key returned from " +
        "'SELECT LAST_INSERT_ID()': " +
        autoIncKeyFromFunc);
} finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
}

```

Example 26.10. Retrieving **AUTO_INCREMENT** column values in **Updatable ResultSets**

```

Statement stmt = null;
ResultSet rs = null;

try {
    //
    // Create a Statement instance that we can use for
    // 'normal' result sets as well as an 'updatable'
    // one, assuming you have a Connection 'conn' to
    // a MySQL database already available
    //

    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
        java.sql.ResultSet.CONCUR_UPDATABLE);

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");

```

```

stmt.executeUpdate(
    "CREATE TABLE autoIncTutorial ("
    + "priKey INT NOT NULL AUTO_INCREMENT, "
    + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

//
// Example of retrieving an AUTO INCREMENT key
// from an updatable result set
//

rs = stmt.executeQuery("SELECT priKey, dataField "
    + "FROM autoIncTutorial");

rs.moveToInsertRow();

rs.updateString("dataField", "AUTO INCREMENT here?");
rs.insertRow();

//
// the driver adds rows at the end
//

rs.last();

//
// We should now be on the row we just inserted
//

int autoIncKeyFromRS = rs.getInt("priKey");
rs.close();

rs = null;

System.out.println("Key returned for inserted row: "
    + autoIncKeyFromRS);
} finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
}

```

When you run the preceding example code, you should get the following output: Key returned from `getGeneratedKeys()`: 1 Key returned from `SELECT LAST_INSERT_ID()`: 1 Key returned for inserted row: 2 You should be aware, that at times, it can be tricky to use the `SELECT LAST_INSERT_ID()` query, as that function's value is scoped to a connection. So, if some other query happens on the same connection, the value will be overwritten. On the other hand, the `getGeneratedKeys()` method is scoped by the `Statement` instance, so it can be used even if other queries happen on the same connection, but not on the same `Statement` instance.

26.4.5.2. Using Connector/J with J2EE and Other Java Frameworks

This section describes how to use Connector/J in several contexts.

26.4.5.2.1. General J2EE Concepts

This section provides general background on J2EE concepts that pertain to use of Connector/J.

26.4.5.2.1.1. Understanding Connection Pooling

Connection pooling is a technique of creating and managing a pool of connections that are ready for use by any thread that needs them.

This technique of pooling connections is based on the fact that most applications only need a thread to have access to a JDBC connection when they are actively processing a transaction, which usually take only milliseconds to complete. When not processing a transac-

tion, the connection would otherwise sit idle. Instead, connection pooling allows the idle connection to be used by some other thread to do useful work.

In practice, when a thread needs to do work against a MySQL or other database with JDBC, it requests a connection from the pool. When the thread is finished using the connection, it returns it to the pool, so that it may be used by any other threads that want to use it.

When the connection is loaned out from the pool, it is used exclusively by the thread that requested it. From a programming point of view, it is the same as if your thread called `DriverManager.getConnection()` every time it needed a JDBC connection, however with connection pooling, your thread may end up using either a new, or already-existing connection.

Connection pooling can greatly increase the performance of your Java application, while reducing overall resource usage. The main benefits to connection pooling are:

- Reduced connection creation time

Although this is not usually an issue with the quick connection setup that MySQL offers compared to other databases, creating new JDBC connections still incurs networking and JDBC driver overhead that will be avoided if connections are recycled.

- Simplified programming model

When using connection pooling, each individual thread can act as though it has created its own JDBC connection, allowing you to use straight-forward JDBC programming techniques.

- Controlled resource usage

If you don't use connection pooling, and instead create a new connection every time a thread needs one, your application's resource usage can be quite wasteful and lead to unpredictable behavior under load.

Remember that each connection to MySQL has overhead (memory, CPU, context switches, and so forth) on both the client and server side. Every connection limits how many resources there are available to your application as well as the MySQL server. Many of these resources will be used whether or not the connection is actually doing any useful work!

Connection pools can be tuned to maximize performance, while keeping resource utilization below the point where your application will start to fail rather than just run slower.

Luckily, Sun has standardized the concept of connection pooling in JDBC through the JDBC-2.0 Optional interfaces, and all major application servers have implementations of these APIs that work fine with MySQL Connector/J.

Generally, you configure a connection pool in your application server configuration files, and access it via the Java Naming and Directory Interface (JNDI). The following code shows how you might use a connection pool from an application deployed in a J2EE application server:

Example 26.11. Using a connection pool with a J2EE application server

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.InitialContext;
import javax.sql.DataSource;

public class MyServletJspOrEjb {

    public void doSomething() throws Exception {
        /*
         * Create a JNDI Initial context to be able to
         * lookup the DataSource
         *
         * In production-level code, this should be cached as
         * an instance or static variable, as it can
         * be quite expensive to create a JNDI context.
         *
         * Note: This code only works when you are using servlets
         * or EJBs in a J2EE application server. If you are
         * using connection pooling in standalone Java code, you
         * will have to create/configure datasources using whatever
         * mechanisms your particular connection pooling library
         * provides.
         */
    }
}
```

```

InitialContext ctx = new InitialContext();

/*
 * Lookup the DataSource, which will be backed by a pool
 * that the application server provides. DataSource instances
 * are also a good candidate for caching as an instance
 * variable, as JNDI lookups can be expensive as well.
 */

DataSource ds =
    (DataSource)ctx.lookup("java:comp/env/jdbc/MySQLDB");

/*
 * The following code is what would actually be in your
 * Servlet, JSP or EJB 'service' method...where you need
 * to work with a JDBC connection.
 */

Connection conn = null;
Statement stmt = null;

try {
    conn = ds.getConnection();

    /*
     * Now, use normal JDBC programming to work with
     * MySQL, making sure to close each resource when you're
     * finished with it, which allows the connection pool
     * resources to be recovered as quickly as possible
     */

    stmt = conn.createStatement();
    stmt.execute("SOME SQL QUERY");

    stmt.close();
    stmt = null;

    conn.close();
    conn = null;
} finally {
    /*
     * close any jdbc instances here that weren't
     * explicitly closed during normal code path, so
     * that we don't 'leak' resources...
     */

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlex) {
            // ignore -- as we can't do anything about it here
        }

        stmt = null;
    }

    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException sqlex) {
            // ignore -- as we can't do anything about it here
        }

        conn = null;
    }
}
}

```

As shown in the example above, after obtaining the JNDI InitialContext, and looking up the DataSource, the rest of the code should look familiar to anyone who has done JDBC programming in the past.

The most important thing to remember when using connection pooling is to make sure that no matter what happens in your code (exceptions, flow-of-control, and so forth), connections, and anything created by them (such as statements or result sets) are closed, so that they may be re-used, otherwise they will be stranded, which in the best case means that the MySQL server resources they represent (such as buffers, locks, or sockets) may be tied up for some time, or worst case, may be tied up forever.

What's the Best Size for my Connection Pool?

As with all other configuration rules-of-thumb, the answer is: it depends. Although the optimal size depends on anticipated load and average database transaction time, the optimum connection pool size is smaller than you might expect. If you take Sun's Java Petstore blueprint application for example, a connection pool of 15-20 connections can serve a relatively moderate load (600 concurrent users)

using MySQL and Tomcat with response times that are acceptable.

To correctly size a connection pool for your application, you should create load test scripts with tools such as Apache JMeter or The Grinder, and load test your application.

An easy way to determine a starting point is to configure your connection pool's maximum number of connections to be unbounded, run a load test, and measure the largest amount of concurrently used connections. You can then work backward from there to determine what values of minimum and maximum pooled connections give the best performance for your particular application.

26.4.5.2.2. Using Connector/J with Tomcat

The following instructions are based on the instructions for Tomcat-5.x, available at <http://jakarta.apache.org/tomcat/tomcat-5.0-doc/jndi-datasource-examples-howto.html> which is current at the time this document was written.

First, install the .jar file that comes with Connector/J in `$CATALINA_HOME/common/lib` so that it is available to all applications installed in the container.

Next, Configure the JNDI DataSource by adding a declaration resource to `$CATALINA_HOME/conf/server.xml` in the context that defines your web application:

```
<Context ....>
...
<Resource name="jdbc/MySQLDB"
          auth="Container"
          type="javax.sql.DataSource"/>
<!-- The name you used above, must match _exactly_ here!
      The connection pool will be bound into JNDI with the name
      "java:/comp/env/jdbc/MySQLDB"
-->
<ResourceParams name="jdbc/MySQLDB">
  <parameter>
    <name>factory</name>
    <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
  </parameter>
  <!-- Don't set this any higher than max_connections on your
        MySQL server, usually this should be a 10 or a few 10's
        of connections, not hundreds or thousands -->
  <parameter>
    <name>maxActive</name>
    <value>10</value>
  </parameter>
  <!-- You don't want to many idle connections hanging around
        if you can avoid it, only enough to soak up a spike in
        the load -->
  <parameter>
    <name>maxIdle</name>
    <value>5</value>
  </parameter>
  <!-- Don't use autoReconnect=true, it's going away eventually
        and it's a crutch for older connection pools that couldn't
        test connections. You need to decide whether your application
        is supposed to deal with SQLExceptions (hint, it should), and
        how much of a performance penalty you're willing to pay
        to ensure 'freshness' of the connection -->
  <parameter>
    <name>validationQuery</name>
    <value>SELECT 1</value>
  </parameter>
  <!-- The most conservative approach is to test connections
        before they're given to your application. For most applications
        this is okay, the query used above is very small and takes
        no real server resources to process, other than the time used
        to traverse the network.
        If you have a high-load application you'll need to rely on
        something else. -->
  <parameter>
    <name>testOnBorrow</name>
```

```

    <value>true</value>
  </parameter>

  <!-- Otherwise, or in addition to testOnBorrow, you can test
    while connections are sitting idle -->

  <parameter>
    <name>testWhileIdle</name>
    <value>true</value>
  </parameter>

  <!-- You have to set this value, otherwise even though
    you've asked connections to be tested while idle,
    the idle evictor thread will never run -->

  <parameter>
    <name>timeBetweenEvictionRunsMillis</name>
    <value>10000</value>
  </parameter>

  <!-- Don't allow connections to hang out idle too long,
    never longer than what wait_timeout is set to on the
    server...A few minutes or even fraction of a minute
    is sometimes okay here, it depends on your application
    and how much spikey load it will see -->

  <parameter>
    <name>minEvictableIdleTimeMillis</name>
    <value>60000</value>
  </parameter>

  <!-- Username and password used when connecting to MySQL -->

  <parameter>
    <name>username</name>
    <value>someuser</value>
  </parameter>

  <parameter>
    <name>password</name>
    <value>somepass</value>
  </parameter>

  <!-- Class name for the Connector/J driver -->

  <parameter>
    <name>driverClassName</name>
    <value>com.mysql.jdbc.Driver</value>
  </parameter>

  <!-- The JDBC connection url for connecting to MySQL, notice
    that if you want to pass any other MySQL-specific parameters
    you should pass them here in the URL, setting them using the
    parameter tags above will have no effect, you will also
    need to use & to separate parameter values as the
    ampersand is a reserved character in XML -->

  <parameter>
    <name>url</name>
    <value>jdbc:mysql://localhost:3306/test</value>
  </parameter>
</ResourceParams>
</Context>

```

In general, you should follow the installation instructions that come with your version of Tomcat, as the way you configure datasources in Tomcat changes from time-to-time, and unfortunately if you use the wrong syntax in your XML file, you will most likely end up with an exception similar to the following:

```

Error: java.sql.SQLException: Cannot load JDBC driver class 'null ' SQL
state: null

```

26.4.5.2.3. Using Connector/J with JBoss

These instructions cover JBoss-4.x. To make the JDBC driver classes available to the application server, copy the .jar file that comes with Connector/J to the [lib](#) directory for your server configuration (which is usually called [default](#)). Then, in the same configuration directory, in the subdirectory named `deploy`, create a datasource configuration file that ends with "-ds.xml", which tells JBoss to deploy this file as a JDBC Datasource. The file should have the following contents:

```

<datasources>
  <local-tx-datasource>
    <!-- This connection pool will be bound into JNDI with the name
      "java:/MySQLDB" -->

```



```

<jndi-name>MySQLDB</jndi-name>
<connection-url>jdbc:mysql://localhost:3306/dbname</connection-url>
<driver-class>com.mysql.jdbc.Driver</driver-class>
<user-name>user</user-name>
<password>pass</password>

<min-pool-size>5</min-pool-size>

<!-- Don't set this any higher than max_connections on your
MySQL server, usually this should be a 10 or a few 10's
of connections, not hundreds or thousands -->

<max-pool-size>20</max-pool-size>

<!-- Don't allow connections to hang out idle too long,
never longer than what wait_timeout is set to on the
server...A few minutes is usually okay here,
it depends on your application
and how much spikey load it will see -->

<idle-timeout-minutes>5</idle-timeout-minutes>

<!-- If you're using Connector/J 3.1.8 or newer, you can use
our implementation of these to increase the robustness
of the connection pool. -->

<exception-sorter-class-name>
com.mysql.jdbc.integration.jboss.ExtendedMysqlExceptionSorter
</exception-sorter-class-name>
<valid-connection-checker-class-name>
com.mysql.jdbc.integration.jboss.MysqlValidConnectionChecker
</valid-connection-checker-class-name>

</local-tx-datasource>
</datasources>

```

26.4.5.2.4. Using Connector/J with Spring

The Spring Framework is a Java-based application framework designed for assisting in application design by providing a way to configure components. The technique used by Spring is a well known design pattern called Dependency Injection (see [Inversion of Control Containers and the Dependency Injection pattern](#)). This article will focus on Java-oriented access to MySQL databases with Spring 2.0. For those wondering, there is a .NET port of Spring appropriately named Spring.NET.

Spring is not only a system for configuring components, but also includes support for aspect oriented programming (AOP). This is one of the main benefits and the foundation for Spring's resource and transaction management. Spring also provides utilities for integrating resource management with JDBC and Hibernate.

For the examples in this section the MySQL world sample database will be used. The first task is to setup a MySQL data source through Spring. Components within Spring use the "bean" terminology. For example, to configure a connection to a MySQL server supporting the world sample database you might use:

```

<util:map id="dbProps">
  <entry key="db.driver" value="com.mysql.jdbc.Driver"/>
  <entry key="db.jdbcurl" value="jdbc:mysql://localhost/world"/>
  <entry key="db.username" value="myuser"/>
  <entry key="db.password" value="mypass"/>
</util:map>

```

In the above example we are assigning values to properties that will be used in the configuration. For the datasource configuration:

```

<bean id="dataSource"
  class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="${db.driver}"/>
  <property name="url" value="${db.jdbcurl}"/>
  <property name="username" value="${db.username}"/>
  <property name="password" value="${db.password}"/>
</bean>

```

The placeholders are used to provide values for properties of this bean. This means that you can specify all the properties of the configuration in one place instead of entering the values for each property on each bean. We do, however, need one more bean to pull this all together. The last bean is responsible for actually replacing the placeholders with the property values.

```

<bean
  class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

```

```
<property name="properties" ref="dbProps"/>
</bean>
```

Now that we have our MySQL data source configured and ready to go, we write some Java code to access it. The example below will retrieve three random cities and their corresponding country using the data source we configured with Spring.

```
// Create a new application context. this processes the Spring config
ApplicationContext ctx =
    new ClassPathXmlApplicationContext("ex1appContext.xml");
// Retrieve the data source from the application context
DataSource ds = (DataSource) ctx.getBean("dataSource");
// Open a database connection using Spring's DataSourceUtils
Connection c = DataSourceUtils.getConnection(ds);
try {
    // retrieve a list of three random cities
    PreparedStatement ps = c.prepareStatement(
        "select City.Name as 'City', Country.Name as 'Country' " +
        "from City inner join Country on City.CountryCode = Country.Code " +
        "order by rand() limit 3");
    ResultSet rs = ps.executeQuery();
    while(rs.next()) {
        String city = rs.getString("City");
        String country = rs.getString("Country");
        System.out.printf("The city %s is in %s\n", city, country);
    }
} catch (SQLException ex) {
    // something has failed and we print a stack trace to analyse the error
    ex.printStackTrace();
    // ignore failure closing connection
    try { c.close(); } catch (SQLException e) { }
} finally {
    // properly release our connection
    DataSourceUtils.releaseConnection(c, ds);
}
```

This is very similar to normal JDBC access to MySQL with the main difference being that we are using `DataSourceUtils` instead of the `DriverManager` to create the connection.

While it may seem like a small difference, the implications are somewhat far reaching. Spring manages this resource in a way similar to a container managed data source in a J2EE application server. When a connection is opened, it can be subsequently accessed in other parts of the code if it is synchronized with a transaction. This makes it possible to treat different parts of your application as transactional instead of passing around a database connection.

26.4.5.2.4.1. Using `JdbcTemplate`

Spring makes extensive use of the Template method design pattern (see [Template Method Pattern](#)). Our immediate focus will be on the `JdbcTemplate` and related classes, specifically `NamedParameterJdbcTemplate`. The template classes handle obtaining and releasing a connection for data access when one is needed.

The next example shows how to use `NamedParameterJdbcTemplate` inside of a DAO (Data Access Object) class to retrieve a random city given a country code.

```
public class Ex2JdbcDao {
    /**
     * Data source reference which will be provided by Spring.
     */
    private DataSource dataSource;

    /**
     * Our query to find a random city given a country code. Notice
     * the ":country" parameter towards the end. This is called a
     * named parameter.
     */
    private String queryString = "select Name from City " +
        "where CountryCode = :country order by rand() limit 1";

    /**
     * Retrieve a random city using Spring JDBC access classes.
     */
    public String getRandomCityByCountryCode(String centryCode) {
        // A template that allows using queries with named parameters
        NamedParameterJdbcTemplate template =
            new NamedParameterJdbcTemplate(dataSource);
        // A java.util.Map is used to provide values for the parameters
        Map params = new HashMap();
        params.put("country", centryCode);
        // We query for an Object and specify what class we are expecting
        return (String)template.queryForObject(queryString, params, String.class);
    }
}
```

```

/**
 * A JavaBean setter-style method to allow Spring to inject the data source.
 * @param dataSource
 */
public void setDataSource(DataSource dataSource) {
    this.dataSource = dataSource;
}
}

```

The focus in the above code is on the `getRandomCityByCountryCode()` method. We pass a country code and use the `Named-ParameterJdbcTemplate` to query for a city. The country code is placed in a Map with the key "country", which is the parameter is named in the SQL query.

To access this code, you need to configure it with Spring by providing a reference to the data source.

```

<bean id="dao" class="code.Ex2JdbcDao">
  <property name="dataSource" ref="dataSource"/>
</bean>

```

At this point, we can just grab a reference to the DAO from Spring and call `getRandomCityByCountryCode()`.

```

// Create the application context
ApplicationContext ctx =
    new ClassPathXmlApplicationContext("ex2appContext.xml");
// Obtain a reference to our DAO
Ex2JdbcDao dao = (Ex2JdbcDao) ctx.getBean("dao");

String countryCode = "USA";

// Find a few random cities in the US
for(int i = 0; i < 4; ++i)
    System.out.printf("A random city in %s is %s\n", countryCode,
        dao.getRandomCityByCountryCode(countryCode));

```

This example shows how to use Spring's JDBC classes to completely abstract away the use of traditional JDBC classes including `Connection` and `PreparedStatement`.

26.4.5.2.4.2. Transactional JDBC Access

You might be wondering how we can add transactions into our code if we don't deal directly with the JDBC classes. Spring provides a transaction management package that not only replaces JDBC transaction management, but also allows declarative transaction management (configuration instead of code).

In order to use transactional database access, we will need to change the storage engine of the tables in the world database. The downloaded script explicitly creates MyISAM tables which do not support transactional semantics. The InnoDB storage engine does support transactions and this is what we will be using. We can change the storage engine with the following statements.

```

ALTER TABLE City ENGINE=InnoDB;
ALTER TABLE Country ENGINE=InnoDB;
ALTER TABLE CountryLanguage ENGINE=InnoDB;

```

A good programming practice emphasized by Spring is separating interfaces and implementations. What this means is that we can create a Java interface and only use the operations on this interface without any internal knowledge of what the actual implementation is. We will let Spring manage the implementation and with this it will manage the transactions for our implementation.

First you create a simple interface:

```

public interface Ex3Dao {
    Integer createCity(String name, String countryCode,
        String district, Integer population);
}

```

This interface contains one method that will create a new city record in the database and return the id of the new record. Next you need to create an implementation of this interface.

```

public class Ex3DaoImpl implements Ex3Dao {
    protected DataSource dataSource;
    protected SqlUpdate updateQuery;
    protected SqlFunction idQuery;
}

```

```

    public Integer createCity(String name, String countryCode,
        String district, Integer population) {
        updateQuery.update(new Object[] { name, countryCode,
            district, population });
        return getLastId();
    }

    protected Integer getLastId() {
        return idQuery.run();
    }
}

```

You can see that we only operate on abstract query objects here and don't deal directly with the JDBC API. Also, this is the complete implementation. All of our transaction management will be dealt with in the configuration. To get the configuration started, we need to create the DAO.

```

<bean id="dao" class="code.Ex3DaoImpl">
    <property name="dataSource" ref="dataSource"/>
    <property name="updateQuery">...</property>
    <property name="idQuery">...</property>
</bean>

```

Now you need to setup the transaction configuration. The first thing you must do is create transaction manager to manage the data source and a specification of what transaction properties are required for for the [dao](#) methods.

```

<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>

<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="*" />
    </tx:attributes>
</tx:advice>

```

The preceding code creates a transaction manager that handles transactions for the data source provided to it. The [txAdvice](#) uses this transaction manager and the attributes specify to create a transaction for all methods. Finally you need to apply this advice with an AOP pointcut.

```

<aop:config>
    <aop:pointcut id="daoMethods"
        expression="execution(* code.Ex3Dao.*(..))"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="daoMethods"/>
</aop:config>

```

This basically says that all methods called on the [Ex3Dao](#) interface will be wrapped in a transaction. To make use of this, you only have to retrieve the [dao](#) from the application context and call a method on the [dao](#) instance.

```

Ex3Dao dao = (Ex3Dao) ctx.getBean("dao");
Integer id = dao.createCity(name, countryCode, district, pop);

```

We can verify from this that there is no transaction management happening in our Java code and it's all configured with Spring. This is a very powerful notion and regarded as one of the most beneficial features of Spring.

26.4.5.2.4.3. Connection Pooling

In many situations, such as web applications, there will be a large number of small database transactions. When this is the case, it usually makes sense to create a pool of database connections available for web requests as needed. Although MySQL does not spawn an extra process when a connection is made, there is still a small amount of overhead to create and setup the connection. Pooling of connections also alleviates problems such as collecting large amounts of sockets in the [TIME_WAIT](#) state.

Setting up pooling of MySQL connections with Spring is as simple as changing the data source configuration in the application context. There are a number of configurations that we can use. The first example is based on the [Jakarta Commons DBCP library](#). The example below replaces the source configuration that was based on [DriverManagerDataSource](#) with DBCP's [BasicDataSource](#).

```

<bean id="dataSource" destroy-method="close"
    class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="${db.driver}"/>
    <property name="url" value="${db.jdbcurl}"/>
    <property name="username" value="${db.username}"/>
    <property name="password" value="${db.password}"/>
    <property name="initialSize" value="3"/>
</bean>

```

```
</bean>
```

The configuration of the two solutions is very similar. The difference is that DBCP will pool connections to the database instead of creating a new connection every time one is requested. We have also set a parameter here called `initialSize`. This tells DBCP that we want three connections in the pool when it is created.

Another way to configure connection pooling is to configure a data source in our J2EE application server. Using JBoss as an example, you can set up the MySQL connection pool by creating a file called `mysql-local-ds.xml` and placing it in the server/default/deploy directory in JBoss. Once we have this setup, we can use JNDI to look it up. With Spring, this lookup is very simple. The data source configuration looks like this.

```
<jee:jndi-lookup id="dataSource" jndi-name="java:MySQL_DS"/>
```

26.4.5.3. Common Problems and Solutions

There are a few issues that seem to be commonly encountered often by users of MySQL Connector/J. This section deals with their symptoms, and their resolutions.

Questions

- [27.4.5.3.1](#): When I try to connect to the database with MySQL Connector/J, I get the following exception:

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

What's going on? I can connect just fine with the MySQL command-line client.

- [27.4.5.3.2](#): My application throws an SQLException 'No Suitable Driver'. Why is this happening?
- [27.4.5.3.3](#): I'm trying to use MySQL Connector/J in an applet or application and I get an exception similar to:

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
are trying to connect to?

(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

- [27.4.5.3.4](#): I have a servlet/application that works fine for a day, and then stops working overnight
- [27.4.5.3.5](#): I'm trying to use JDBC-2.0 updatable result sets, and I get an exception saying my result set is not updatable.
- [27.4.5.3.6](#): I cannot connect to the MySQL server using Connector/J, and I'm sure the connection parameters are correct.
- [27.4.5.3.7](#): I am trying to connect to my MySQL server within my application, but I get the following error and stack trace:

```
java.net.SocketException
MESSAGE: Software caused connection abort: recv failed

STACKTRACE:

java.net.SocketException: Software caused connection abort: recv failed
at java.net.SocketInputStream.socketRead0(Native Method)
at java.net.SocketInputStream.read(Unknown Source)
at com.mysql.jdbc.MySQLIO.readFully(MySQLIO.java:1392)
at com.mysql.jdbc.MySQLIO.readPacket(MySQLIO.java:1414)
at com.mysql.jdbc.MySQLIO.doHandshake(MySQLIO.java:625)
at com.mysql.jdbc.Connection.createNewIO(Connection.java:1926)
at com.mysql.jdbc.Connection.<init>(Connection.java:452)
at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:411)
```

- [27.4.5.3.8](#): My application is deployed through JBoss and I am using transactions to handle the statements on the MySQL database. Under heavy loads I am getting an error and stack trace, but these only occur after a fixed period of heavy activity.
- [27.4.5.3.9](#): When using `gcj` an `java.io.CharConversionException` is raised when working with certain character sequences.

- [27.4.5.3.10](#): Updating a table that contains a primary key that is either `FLOAT` or compound primary key that uses `FLOAT` fails to update the table and raises an exception.

Questions and Answers

27.4.5.3.1: When I try to connect to the database with MySQL Connector/J, I get the following exception:

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

What's going on? I can connect just fine with the MySQL command-line client.

MySQL Connector/J must use TCP/IP sockets to connect to MySQL, as Java does not support Unix Domain Sockets. Therefore, when MySQL Connector/J connects to MySQL, the security manager in MySQL server will use its grant tables to determine whether the connection should be allowed.

You must add the necessary security credentials to the MySQL server for this to happen, using the `GRANT` statement to your MySQL Server. See [Section 12.5.1.3, “GRANT Syntax”](#), for more information.

Note

Testing your connectivity with the `mysql` command-line client will not work unless you add the `--host` flag, and use something other than `localhost` for the host. The `mysql` command-line client will use Unix domain sockets if you use the special hostname `localhost`. If you are testing connectivity to `localhost`, use `127.0.0.1` as the hostname instead.

Warning

Changing privileges and permissions improperly in MySQL can potentially cause your server installation to not have optimal security properties.

27.4.5.3.2: My application throws an SQLException 'No Suitable Driver'. Why is this happening?

There are three possible causes for this error:

- The Connector/J driver is not in your `CLASSPATH`, see [Section 26.4.2, “Connector/J Installation”](#).
- The format of your connection URL is incorrect, or you are referencing the wrong JDBC driver.
- When using DriverManager, the `jdbc.drivers` system property has not been populated with the location of the Connector/J driver.

27.4.5.3.3: I'm trying to use MySQL Connector/J in an applet or application and I get an exception similar to:

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
are trying to connect to?

(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

Either you're running an Applet, your MySQL server has been installed with the `--skip-networking` option set, or your MySQL server has a firewall sitting in front of it.

Applets can only make network connections back to the machine that runs the web server that served the `.class` files for the applet. This means that MySQL must run on the same machine (or you must have some sort of port re-direction) for this to work. This also means that you will not be able to test applets from your local file system, you must always deploy them to a web server.

MySQL Connector/J can only communicate with MySQL using TCP/IP, as Java does not support Unix domain sockets. TCP/IP communication with MySQL might be affected if MySQL was started with the `--skip-networking` flag, or if it is firewalled.

If MySQL has been started with the `--skip-networking` option set (the Debian Linux package of MySQL server does this for example), you need to comment it out in the file `/etc/mysql/my.cnf` or `/etc/my.cnf`. Of course your `my.cnf` file might also exist in the `data` direct-

ory of your MySQL server, or anywhere else (depending on how MySQL was compiled for your system). Binaries created by MySQL AB always look in `/etc/my.cnf` and `[datadir]/my.cnf`. If your MySQL server has been firewalled, you will need to have the firewall configured to allow TCP/IP connections from the host where your Java code is running to the MySQL server on the port that MySQL is listening to (by default, 3306).

27.4.5.3.4: I have a servlet/application that works fine for a day, and then stops working overnight

MySQL closes connections after 8 hours of inactivity. You either need to use a connection pool that handles stale connections or use the "autoReconnect" parameter (see [Section 26.4.4.1, "Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J"](#)).

Also, you should be catching `SQLExceptions` in your application and dealing with them, rather than propagating them all the way until your application exits, this is just good programming practice. MySQL Connector/J will set the `SQLState` (see [`java.sql.SQLException.getSQLState\(\)`](#) in your APIDocs) to "08S01" when it encounters network-connectivity issues during the processing of a query. Your application code should then attempt to re-connect to MySQL at this point.

The following (simplistic) example shows what code that can handle these exceptions might look like:

Example 26.12. Example of transaction with retry logic

```
public void doBusinessOp() throws SQLException {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    //
    // How many times do you want to retry the transaction
    // (or at least _getting_ a connection)?
    //
    int retryCount = 5;

    boolean transactionCompleted = false;

    do {
        try {
            conn = getConnection(); // assume getting this from a
                                   // javax.sql.DataSource, or the
                                   // java.sql.DriverManager

            conn.setAutoCommit(false);

            //
            // Okay, at this point, the 'retry-ability' of the
            // transaction really depends on your application logic,
            // whether or not you're using autocommit (in this case
            // not), and whether you're using transactional storage
            // engines
            //
            // For this example, we'll assume that it's _not_ safe
            // to retry the entire transaction, so we set retry
            // count to 0 at this point
            //
            // If you were using exclusively transaction-safe tables,
            // or your application could recover from a connection going
            // bad in the middle of an operation, then you would not
            // touch 'retryCount' here, and just let the loop repeat
            // until retryCount == 0.
            //
            retryCount = 0;

            stmt = conn.createStatement();

            String query = "SELECT foo FROM bar ORDER BY baz";

            rs = stmt.executeQuery(query);

            while (rs.next()) {
            }

            rs.close();
            rs = null;

            stmt.close();
            stmt = null;

            conn.commit();
            conn.close();
            conn = null;

            transactionCompleted = true;
        } catch (SQLException sqlEx) {
```

```

//
// The two SQL states that are 'retry-able' are 08S01
// for a communications error, and 40001 for deadlock.
//
// Only retry if the error was due to a stale connection,
// communications problem or deadlock
//
String sqlState = sqlEx.getSQLState();

if ("08S01".equals(sqlState) || "40001".equals(sqlState)) {
    retryCount--;
} else {
    retryCount = 0;
}
} finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) {
            // You'd probably want to log this . . .
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) {
            // You'd probably want to log this as well . . .
        }
    }

    if (conn != null) {
        try {
            //
            // If we got here, and conn is not null, the
            // transaction should be rolled back, as not
            // all work has been done

            try {
                conn.rollback();
            } finally {
                conn.close();
            }
        } catch (SQLException sqlEx) {
            //
            // If we got an exception here, something
            // pretty serious is going on, so we better
            // pass it up the stack, rather than just
            // logging it. . .
            throw sqlEx;
        }
    }
}
} while (!transactionCompleted && (retryCount > 0));
}

```

Note

Use of the [autoReconnect](#) option is not recommended because there is no safe method of reconnecting to the MySQL server without risking some corruption of the connection state or database state information. Instead, you should use a connection pool which will enable your application to connect to the MySQL server using an available connection from the pool. The [autoReconnect](#) facility is deprecated, and may be removed in a future release.

27.4.5.3.5: I'm trying to use JDBC-2.0 updatable result sets, and I get an exception saying my result set is not updatable.

Because MySQL does not have row identifiers, MySQL Connector/J can only update result sets that have come from queries on tables that have at least one primary key, the query must select every primary key and the query can only span one table (that is, no joins). This is outlined in the JDBC specification.

Note that this issue only occurs when using updatable result sets, and is caused because Connector/J is unable to guarantee that it can identify the correct rows within the result set to be updated without having a unique reference to each row. There is no requirement to have a unique field on a table if you are using [UPDATE](#) or [DELETE](#) statements on a table where you can individually specify the criteria to be matched using a [WHERE](#) clause.

27.4.5.3.6: I cannot connect to the MySQL server using Connector/J, and I'm sure the connection parameters are correct.

Make sure that the [skip-networking](#) option has not been enabled on your server. Connector/J must be able to communicate with your server over TCP/IP, named sockets are not supported. Also ensure that you are not filtering connections through a Firewall or other network security system. For more information, see [Section B.1.2.2, “Can't connect to \[local\] MySQL server”](#).

27.4.5.3.7: I am trying to connect to my MySQL server within my application, but I get the following error and stack trace:

```
java.net.SocketException
MESSAGE: Software caused connection abort: recv failed

STACKTRACE:

java.net.SocketException: Software caused connection abort: recv failed
at java.net.SocketInputStream.socketRead0(Native Method)
at java.net.SocketInputStream.read(Unknown Source)
at com.mysql.jdbc.MySQLIO.readFully(MySQLIO.java:1392)
at com.mysql.jdbc.MySQLIO.readPacket(MySQLIO.java:1414)
at com.mysql.jdbc.MySQLIO.doHandshake(MySQLIO.java:625)
at com.mysql.jdbc.Connection.createNewIO(Connection.java:1926)
at com.mysql.jdbc.Connection.<init>(Connection.java:452)
at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:411)
```

The error probably indicates that you are using an older version of the Connector/J JDBC driver (2.0.14 or 3.0.x) and you are trying to connect to a MySQL server with version 4.1.x or newer. The older drivers are not compatible with 4.1 or newer of MySQL as they do not support the newer authentication mechanisms.

It is likely that the older version of the Connector/J driver exists within your application directory or your [CLASSPATH](#) includes the older Connector/J package.

27.4.5.3.8: My application is deployed through JBoss and I am using transactions to handle the statements on the MySQL database. Under heavy loads I am getting an error and stack trace, but these only occur after a fixed period of heavy activity.

This is a JBoss, not Connector/J, issue and is connected to the use of transactions. Under heavy loads the time taken for transactions to complete can increase, and the error is caused because you have exceeded the predefined timeout.

You can increase the timeout value by setting the [TransactionTimeout](#) attribute to the [TransactionManagerService](#) within the `/conf/jboss-service.xml` file (pre-4.0.3) or `/deploy/jta-service.xml` for JBoss 4.0.3 or later. See [Transaction-Timeout](#) within the JBoss wiki for more information.

27.4.5.3.9: When using gcj an java.io.CharConversionException is raised when working with certain character sequences.

This is a known issue with [gcj](#) which raises an exception when it reaches an unknown character or one it cannot convert. You should add `useJvmCharsetConverters=true` to your connection string to force character conversion outside of the [gcj](#) libraries, or try a different JDK.

27.4.5.3.10: Updating a table that contains a primary key that is either FLOAT or compound primary key that uses FLOAT fails to update the table and raises an exception.

Connector/J adds conditions to the [WHERE](#) clause during an [UPDATE](#) to check the old values of the primary key. If there is no match then Connector/J considers this a failure condition and raises an exception.

The problem is that rounding differences between supplied values and the values stored in the database may mean that the values never match, and hence the update fails. The issue will affect all queries, not just those from Connector/J.

To prevent this issue, use a primary key that does not use [FLOAT](#). If you have to use a floating point column in your primary key use [DOUBLE](#) or [DECIMAL](#) types in place of [FLOAT](#).

26.4.6. Connector/J Support

26.4.6.1. Connector/J Community Support

MySQL AB provides assistance to the user community by means of its mailing lists. For Connector/J related issues, you can get help from experienced users by using the MySQL and Java mailing list. Archives and subscription information is available online at <http://lists.mysql.com/java>.

For information about subscribing to MySQL mailing lists or to browse list archives, visit <http://lists.mysql.com/>. See [Section 1.7.1, “MySQL Mailing Lists”](#).

Community support from experienced users is also available through the [JDBC Forum](#). You may also find help from other users in the other MySQL Forums, located at <http://forums.mysql.com>. See [Section 1.7.2](#), “MySQL Community Support at the MySQL Forums”.

26.4.6.2. How to Report Connector/J Bugs or Problems

The normal place to report bugs is <http://bugs.mysql.com/>, which is the address for our bugs database. This database is public, and can be browsed and searched by anyone. If you log in to the system, you will also be able to enter new reports.

If you have found a sensitive security bug in MySQL, you can send email to security_at_mysql.com.

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release.

This section will help you write your report correctly so that you don't waste your time doing things that may not help us much or at all.

If you have a repeatable bug report, please report it to the bugs database at <http://bugs.mysql.com/>. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

To report other problems, you can use one of the MySQL mailing lists.

Remember that it is possible for us to respond to a message containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details don't matter.

A good principle is this: If you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of Connector/J or MySQL used, and (b) not fully describing the platform on which Connector/J is installed (including the JVM version, and the platform type and version number that MySQL itself is installed on).

This is highly relevant information, and in 99 cases out of 100, the bug report is useless without it. Very often we get questions like, “Why doesn't this work for me?” Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has already been fixed in newer MySQL versions.

Sometimes the error is platform-dependent; in such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If at all possible, you should create a repeatable, standalone testcase that doesn't involve any third-party classes.

To streamline this process, we ship a base class for testcases with Connector/J, named `'com.mysql.jdbc.util.BaseBugReport'`. To create a testcase for Connector/J using this class, create your own class that inherits from `com.mysql.jdbc.util.BaseBugReport` and override the methods `setUp()`, `tearDown()` and `runTest()`.

In the `setUp()` method, create code that creates your tables, and populates them with any data needed to demonstrate the bug.

In the `runTest()` method, create code that demonstrates the bug using the tables and data you created in the `setUp` method.

In the `tearDown()` method, drop any tables you created in the `setUp()` method.

In any of the above three methods, you should use one of the variants of the `getConnection()` method to create a JDBC connection to MySQL:

- `getConnection()` - Provides a connection to the JDBC URL specified in `getUrl()`. If a connection already exists, that connection is returned, otherwise a new connection is created.
- `getNewConnection()` - Use this if you need to get a new connection for your bug report (i.e. there's more than one connection involved).
- `getConnection(String url)` - Returns a connection using the given URL.
- `getConnection(String url, Properties props)` - Returns a connection using the given URL and properties.

If you need to use a JDBC URL that is different from `'jdbc:mysql:///test'`, override the method `getUrl()` as well.

Use the `assertTrue(boolean expression)` and `assertTrue(String failureMessage, boolean expression)` methods to create conditions that must be met in your testcase demonstrating the behavior you are expecting (vs. the behavior you are observing, which is why you are most likely filing a bug report).

Finally, create a `main()` method that creates a new instance of your testcase, and calls the `run` method:

```
public static void main(String[] args) throws Exception {
    new MyBugReport().run();
}
```

Once you have finished your testcase, and have verified that it demonstrates the bug you are reporting, upload it with your bug report to <http://bugs.mysql.com/>.

26.4.6.3. Connector/J Change History

The Connector/J Change History (Changelog) is located with the main Changelog for MySQL. See [Section C.5, “MySQL Connector/J Change History”](#).

26.5. MySQL Connector/MXJ

MySQL Connector/MXJ is a solution for deploying the MySQL database engine (`mysqld`) intelligently from within a Java package.

26.5.1. Introduction to Connector/MXJ

MySQL Connector/MXJ is a Java Utility package for deploying and managing a MySQL database. Deploying and using MySQL can be as easy as adding an additional parameter to the JDBC connection url, which will result in the database being started when the first connection is made. This makes it easy for Java developers to deploy applications which require a database by reducing installation barriers for their end-users.

MySQL Connector/MXJ makes the MySQL database appear to be a java-based component. It does this by determining what platform the system is running on, selecting the appropriate binary, and launching the executable. It will also optionally deploy an initial database, with any specified parameters.

Included are instructions for use with a JDBC driver and deploying as a JMX MBean to JBoss.

You can download sources and binaries from: <http://dev.mysql.com/downloads/connector/mxj/>

This a beta release and feedback is welcome and encouraged.

Please send questions or comments to the [MySQL and Java mailing list](#).

26.5.1.1. Connector/MXJ Versions

Connector/MX

- Connector/MXJ 5.x, currently in beta status, includes `mysqld` version 5.x and includes binaries for Linux x86, Mac OS X PPC, Windows XP/NT/2000 x86 and Solaris SPARC. Connector/MXJ 5.x requires the Connector/J 5.x package.

The exact version of `mysqld` included depends on the version of Connector/MXJ

- Connector/MXJ v5.0.3 included MySQL v5.0.22
 - Connector/MXJ v5.0.4 includes MySQL v5.0.27 (Community) or MySQL v5.0.32 (Enterprise)
 - Connector/MXJ v5.0.6 includes MySQL 5.0.37 (Community)
- Connector/MXJ 1.x includes `mysqld` version 4.1.13 and includes binaries for Linux x86, Windows XP/NT/2000 x86 and Solaris SPARC. Connector/MXJ 1.x requires the Connector/J 3.x package.

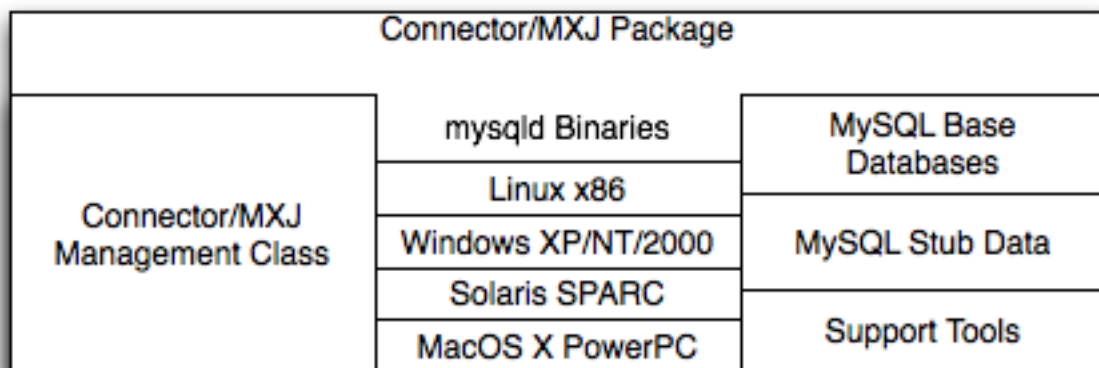
A summary of the different MySQL versions supplied with each Connector/MXJ release are shown in the table.

Connector/MXJ Version	MySQL Version(s)
5.0.6	5.0.37 (CS), 5.0.40 (ES)
5.0.5	5.0.37 (CS), 5.0.36 (ES)
5.0.4	5.0.27 (CS), 5.0.32 (ES)
5.0.3	5.0.22
5.0.2	5.0.19

This guide provides information on the Connector/MXJ 5.x release. For information on using the older releases, please see the documentation included with the appropriate distribution.

26.5.1.2. Connector/MXJ Overview

Connector/MXJ consists of a Java class, a copy of the `mysqld` binary for a specific list of platforms, and associated files and support utilities. The Java class controls the initialization of an instance of the embedded `mysqld` binary, and the ongoing management of the `mysqld` process. The entire sequence and management can be controlled entirely from within Java using the Connector/MXJ Java classes. You can see an overview of the contents of the Connector/MXJ package in the figure below.



It is important to note that Connector/MXJ is not an embedded version of MySQL, or a version of MySQL written as part of a Java class. Connector/MXJ works through the use of an embedded, compiled binary of `mysqld` as would normally be used when deploying a standard MySQL installation.

It is the Connector/MXJ wrapper, support classes and tools, that enable Connector/MXJ to appear as a MySQL instance.

When Connector/MXJ is initialized, the corresponding `mysqld` binary for the current platform is extracted, along with a pre-configured data directed. Both are contained within the Connector/MXJ JAR file. The `mysqld` instance is then started, with any additional options as specified during the initialization, and the MySQL database becomes accessible.

Because Connector/MXJ works in combination with Connector/J, you can access and integrate with the MySQL instance through a JDBC connection. When you have finished with the server, the instance is terminated, and, by default, any data created during the session is retained within the temporary directory created when the instance was started.

Connector/MXJ and the embedded `mysqld` instance can be deployed in a number of environments where relying on an existing database, or installing a MySQL instance would be impossible, including CD-ROM embedded database applications and temporary database requirements within a Java-based application environment.

26.5.2. Connector/MXJ Installation

Connector/MXJ does not have an installation application or process, but there are some steps you can follow to make the installation and deployment of Connector/MXJ easier.

Before you start, there are some baseline requirements for

- Java Runtime Environment (v1.4.0 or newer) if you are only going to deploy the package.
- Java Development Kit (v1.4.0 or newer) if you want to build Connector/MXJ from source.
- Connector/J 5.0 or newer.

Depending on your target installation/deployment environment you may also require:

- JBoss - 4.0rc1 or newer
- Apache Tomcat - 5.0 or newer
- Sun's JMX reference implementation version 1.2.1 (from <http://java.sun.com/products/JavaManagement/>)

26.5.2.1. Supported Platforms

Connector/MXJ is compatible with any platform supporting Java and MySQL. By default, Connector/MXJ incorporates the `mysqlld` binary for a select number of platforms, as outlined below.

- Linux, i386
- Windows NT, Windows 2000, Windows XP, x86
- Solaris 8, SPARC 32 (compatible with Solaris 8, Solaris 9 and Solaris 10 on SPARC 32-bit and 64-bit platforms)
- Mac OS X, PowerPC

For more information on packaging your own Connector/MXJ with the platforms you require, see [Section 26.5.5.1, “Creating your own Connector/MXJ Package”](#)

26.5.2.2. Connector/MXJ Base Installation

Because there is no formal installation process, the method, installation directory, and access methods you use for Connector/MXJ are entirely up to your individual requirements.

To perform a basic installation, choose a target directory for the files included in the Connector/MXJ package. On Unix/Linux systems you may opt to use a directory such as `/usr/local/connector-mxj`; On Windows, you may want to install the files in the base directory, `C:\Connector-MXJ`, or within the `Program Files` directory.

To install the files, for a Connector/MXJ 5.0.4 installation:

1. Download the Connector/MXJ package, either in Tar/Gzip format (ideal for Unix/Linux systems) or Zip format (Windows).
2. Extract the files from the package. This will create a directory `connector-mxj`. Copy and optionally rename this directory to your desired location.
3. For best results, you should update your global `CLASSPATH` variable with the location of the required `jar` files.

Within Unix/Linux you can do this globally by editing the global shell profile, or on a user by user basis by editing their individual shell profile.

On Windows 2000, Windows NT and Windows XP, you can edit the global `CLASSPATH` by editing the `Environment Variables` configured through the `System` control panel.

For Connector/MXJ 5.0.4 and later you need the following JAR files in your `CLASSPATH`

1. `connector-mxj.jar` — contains the main Connector/MXJ classes.

2. `connector-mxj-db-files.jar` — contains the embedded `mysqld` and database files.
3. `aspectjrt.jar` — the AspectJ runtime library, located in `lib/aspectjrt.jar` in the Connector/MXJ package.
4. `connector-j.jar` — Connector/J, see [Section 26.4, “MySQL Connector/J”](#).

For Connector/MXJ 5.0.3 and earlier, you need the following JAR files:

1. `connector-mxj.jar`
2. `aspectjrt.jar` — the AspectJ runtime library, located in `lib/aspectjrt.jar` in the Connector/MXJ package.
3. `connector-j.jar` — Connector/J, see [Section 26.4, “MySQL Connector/J”](#).

26.5.2.3. Connector/MXJ Quick Start Guide

Once you have extracted the Connector/MXJ and Connector/J components you can run one of the sample applications that initiates a MySQL instance. You can test the installation by running the [ConnectorMXJUrlTestExample](#):

```
java ConnectorMXJUrlTestExample
jdbc:mysql:mxj://localhost:3336/test?serverbasedir=/var/tmp/test-mxj
[MysqldResource] launching mysqld (getOptions)
[/var/tmp/test-mxj/bin/mysqld][--no-defaults] »
[--pid-file=/var/tmp/test-mxj/data/MysqldResource.pid] »
[--socket=mysql.sock][--datadir=/var/tmp/test-mxj/data] »
[--port=3336][--basedir=/var/tmp/test-mxj]
[MysqldResource] launching mysqld (driver_launched_mysqld_1)
InnoDB: The first specified data file ./ibdata1 did not exist:
InnoDB: a new database to be created!
060726 15:40:42 InnoDB: Setting file ./ibdata1 size to 10 MB
InnoDB: Database physically writes the file full: wait...
060726 15:40:43 InnoDB: Log file ./ib_logfile0 did not exist: new to be created
InnoDB: Setting log file ./ib_logfile0 size to 5 MB
InnoDB: Database physically writes the file full: wait...
060726 15:40:43 InnoDB: Log file ./ib_logfile1 did not exist: new to be created
InnoDB: Setting log file ./ib_logfile1 size to 5 MB
InnoDB: Database physically writes the file full: wait...
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
060726 15:40:44 InnoDB: Started; log sequence number 0 0
060726 15:40:44 [Note] /var/tmp/test-mxj/bin/mysqld: ready for connections.
Version: '5.0.22-max' socket: 'mysql.sock' port: 3336 MySQL Community Edition - Experimental (GPL)
[MysqldResource] mysqld running as process: 1210
-----
5.0.22-max
-----
[MysqldResource] stopping mysqld (process: 1210)
060726 15:40:44 [Note] /var/tmp/test-mxj/bin/mysqld: Normal shutdown

060726 15:40:45 InnoDB: Starting shutdown...
060726 15:40:48 InnoDB: Shutdown completed; log sequence number 0 43655
060726 15:40:48 [Note] /var/tmp/test-mxj/bin/mysqld: Shutdown complete

[MysqldResource] clearing options
[MysqldResource] shutdown complete
```

The above output shows an instance of MySQL starting, the necessary files being created (log files, InnoDB data files) and the MySQL database entering the running state. The instance is then shutdown by Connector/MXJ before the example terminates.

26.5.2.4. Deploying Connector/MXJ using Driver Launch

Connector/MXJ and Connector/J work together to enable you to launch an instance of the `mysqld` server through the use of a keyword in the JDBC connection string. Deploying Connector/MXJ within a Java application can be automated through this method, making the deployment of Connector/MXJ a simple process:

1. Download and unzip Connector/MXJ, add `connector-mxj.jar` to the `CLASSPATH`.

If you are using Connector/MXJ v5.0.4 or later you will also need to add the `connector-mxj-db-files.jar` file to your `CLASSPATH`.

2. To the JDBC connection string, embed the `mxj` keyword, for example: `jdbc:mysql:mxj://localhost:PORT/DBNAME`.

For more details, see [Section 26.5.3, “Connector/MXJ Configuration”](#).

26.5.2.5. Deploying Connector/MXJ within JBoss

For deployment within a JBoss environment, you must configure the JBoss environment to use the Connector/MXJ component within the JDBC parameters:

1. Download Connector/MXJ and copy the `connector-mxj.jar` file to the `$JBOSS_HOME/server/default/lib` directory.

If you are using Connector/MXJ v5.0.4 or later you will also need to copy the `connector-mxj-db-files.jar` file to `$JBOSS_HOME/server/default/lib`.

2. Download Connector/J and copy the `mysql-connector-java-5.0.4-bin.jar` file to the `$JBOSS_HOME/server/default/lib` directory.
3. Create an MBean service xml file in the `$JBOSS_HOME/server/default/deploy` directory with any attributes set, for instance the `datadir` and `autostart`.
4. Set the JDBC parameters of your web application to use:

```
String driver = "com.mysql.jdbc.Driver";
String url = "jdbc:mysql:///test?propertiesTransform="+
    "com.mysql.management.jmx.ConnectorMXJPropertiesTransform";
String user = "root";
String password = "";
Class.forName(driver);
Connection conn = DriverManager.getConnection(url, user, password);
```

You may wish to create a separate users and database table spaces for each application, rather than using "root and test".

We highly suggest having a routine backup procedure for backing up the database files in the `datadir`.

26.5.2.6. Verifying Installation using JUnit

The best way to ensure that your platform is supported is to run the JUnit tests. These will test the Connector/MXJ classes and the associated components.

26.5.2.6.1. JUnit Test Requirements

The first thing to do is make sure that the components will work on the platform. The `MysqldResource` class is really a wrapper for a native version of MySQL, so not all platforms are supported. At the time of this writing, Linux on the i386 architecture has been tested and seems to work quite well, as does OS X v10.3. There has been limited testing on Windows and Solaris.

Requirements:

1. JDK-1.4 or newer (or the JRE if you aren't going to be compiling the source or JSPs).
2. MySQL Connector/J version 5.0 or newer (from <http://dev.mysql.com/downloads/connector/j/>) installed and available via your `CLASSPATH`.
3. The `javax.management` classes for JMX version 1.2.1, these are present in the following application servers:
 - JBoss - 4.0rc1 or newer.
 - Apache Tomcat - 5.0 or newer.

- Sun's JMX reference implementation version 1.2.1 (from <http://java.sun.com/products/JavaManagement/>).
4. JUnit 3.8.1 (from <http://www.junit.org/>).

If building from source, All of the requirements from above, plus:

1. Ant version 1.5 or newer (download from <http://ant.apache.org/>).

26.5.2.6.2. Running the JUnit Tests

1. The tests attempt to launch MySQL on the port 3336. If you have a MySQL running, it may conflict, but this isn't very likely because the default port for MySQL is 3306. However, You may set the "c-mxj_test_port" Java property to a port of your choosing. Alternatively, you may wish to start by shutting down any instances of MySQL you have running on the target machine.

The tests suppress output to the console by default. For verbose output, you may set the "c-mxj_test_silent" Java property to "false".

2. To run the JUnit test suite, the \$CLASSPATH must include the following:
 - JUnit
 - JMX
 - Connector/J
 - MySQL Connector/MXJ
3. If `connector-mxj.jar` is not present in your download, unzip MySQL Connector/MXJ source archive.

```
cd mysqljmx
ant dist
```

Then add `$TEMP/cmjx/stage/connector-mxj/connector-mxj.jar` to the CLASSPATH.

4. if you have `junit`, execute the unit tests. From the command line, type:

```
java junit.textui.TestRunner com.mysql.management.AllTestsSuite
```

The output should look something like this:

```
.....
.....
Time: 259.438
OK (101 tests)
```

Note that the tests are a bit slow near the end, so please be patient.

26.5.3. Connector/MXJ Configuration

26.5.3.1. Running as part of the JDBC Driver

A feature of the MySQL Connector/J JDBC driver is the ability to specify a connection to an embedded Connector/MXJ instance through the use of the `mxj` keyword in the JDBC connection string.

In the following example, we have a program which creates a connection, executes a query, and prints the result to the System.out. The MySQL database will be deployed and started as part of the connection process, and shutdown as part of the finally block.

You can find this file in the Connector/MXJ package as [src/ConnectorMXJUrlTestExample.java](#).

```
import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

import com.mysql.management.driverlaunched.ServerLauncherSocketFactory;

public class ConnectorMXJUrlTestExample {
    public static String DRIVER = "com.mysql.jdbc.Driver";
    public static String JAVA_IO_TMPDIR = "java.io.tmpdir";

    public static void main(String[] args) throws Exception {
        File ourAppDir = new File(System.getProperty(JAVA_IO_TMPDIR));
        File databaseDir = new File(ourAppDir, "test-mxj");
        int port = 3336;

        String url = "jdbc:mysql:mxj://localhost:" + port + "/test" + "?"
            + "serverbasedir=" + databaseDir;

        System.out.println(url);

        String userName = "root";
        String password = "";

        Class.forName(DRIVER);
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url, userName, password);
            printQueryResults(conn, "SELECT VERSION()");
        } finally {
            try {
                if (conn != null)
                    conn.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        ServerLauncherSocketFactory.shutdown(databaseDir, null);
    }

    public static void printQueryResults(Connection conn, String SQLquery)
        throws Exception {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(SQLquery);
        int columns = rs.getMetaData().getColumnCount();
        System.out.println("-----");
        System.out.println();
        while (rs.next()) {
            for (int i = 1; i <= columns; i++) {
                System.out.println(rs.getString(i));
            }
            System.out.println();
        }
        rs.close();
        stmt.close();
        System.out.println("-----");
        System.out.flush();
        Thread.sleep(100); // wait for System.out to finish flush
    }
}
```

To run the above program, be sure to have connector-mxj.jar and Connector/J in the CLASSPATH. Then type:

```
java ConnectorMXJTestExample
```

26.5.3.2. Running within a Java Object

If you have a java application and wish to “embed” a MySQL database, make use of the `com.mysql.management.MysqldResource` class directly. This class may be instantiated with the default (no argument) constructor, or by passing in a `java.io.File` object representing the directory you wish the server to be “unzipped” into. It may also be instantiated with printstreams for “stdout” and “stderr” for logging.

Once instantiated, a `java.util.Map`, the object will be able to provide a `java.util.Map` of server options appropriate for the platform and version of MySQL which you will be using.

The `MysqldResource` enables you to “start” MySQL with a `java.util.Map` of server options which you provide, as well as “shutdown”

the database. The following example shows a simplistic way to embed MySQL in an application using plain java objects.

You can find this file in the Connector/MXJ package as [src/ConnectorMXJObjectTestExample.java](#).

```
import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Map;

import com.mysql.management.MysqldResource;

public class ConnectorMXJObjectTestExample {
    public static String DRIVER = "com.mysql.jdbc.Driver";

    public static String JAVA_IO_TMPDIR = "java.io.tmpdir";

    public static void main(String[] args) throws Exception {
        File ourAppDir = new File(System.getProperty(JAVA_IO_TMPDIR));
        File databaseDir = new File(ourAppDir, "mysql-mxj");
        int port = 3336;

        MysqldResource mysqldResource = startDatabase(databaseDir, port);

        String userName = "root";
        String password = "";

        Class.forName(DRIVER);
        Connection conn = null;
        try {
            String url = "jdbc:mysql://localhost:" + port + "/test";
            conn = DriverManager.getConnection(url, userName, password);
            printQueryResults(conn, "SELECT VERSION()");
        } finally {
            try {
                if (conn != null) {
                    conn.close();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            try {
                mysqldResource.shutdown();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static MysqldResource startDatabase(File databaseDir, int port) {
        MysqldResource mysqldResource = new MysqldResource(databaseDir);

        Map database_options = new HashMap();
        database_options.put("port", Integer.toString(port));
        mysqldResource.start("test-mysqld-thread", database_options);

        if (!mysqldResource.isRunning()) {
            throw new RuntimeException("MySQL did not start.");
        }

        System.out.println("MySQL is running.");

        return mysqldResource;
    }

    public static void printQueryResults(Connection conn, String SQLquery)
        throws Exception {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(SQLquery);
        int columns = rs.getMetaData().getColumnCount();
        System.out.println("-----");
        System.out.println();
        while (rs.next()) {
            for (int i = 1; i <= columns; i++) {
                System.out.println(rs.getString(i));
            }
            System.out.println();
        }
        rs.close();
        stmt.close();
        System.out.println("-----");
        System.out.flush();
        Thread.sleep(100); // wait for System.out to finish flush
    }
}
```

26.5.3.3. Setting server options

Of course there are many options we may wish to set for a MySQL database. These options may be specified as part of the JDBC connection string simply by prefixing each server option with "server.". In the following example we set two driver parameters and two server parameters:

```
String url = "jdbc:mysql://" + hostColonPort + "/"
    + "?"
    + "cacheServerConfiguration=true"
    + "&"
    + "useLocalSessionState=true"
    + "&"
    + "server.basedir=/opt/myapp/db"
    + "&"
    + "server.datadir=/mnt/bigdisk/myapp/data";
```

26.5.4. Connector/MXJ Reference

26.5.4.1. MysqldResource API

26.5.4.1.1. MysqldResource Constructors

The `MysqldResource` class supports three different constructor forms:

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString, PrintStream out, PrintStream err, Utils util)`

The most detailed constructor, enables you to set the base directory, data directory, select a server by its version string, standard out and standard error and MySQL utilities class.

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString, PrintStream out, PrintStream err)`

Enables you to set the base directory, data directory, select a server by its version string, standard out and standard error.

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString)`

Enables you to set the base directory, data directory and select a server by its version string. Output for standard out and standard err are directed to `System.out` and `System.err`.

- `public MysqldResource(File baseDir, File dataDir)`

Enables you to set the base directory and data directory. The default MySQL version is selected, and output for standard out and standard err are directed to `System.out` and `System.err`.

- `public MysqldResource(File baseDir);`

Allows the setting of the "basedir" to deploy the MySQL files to. Output for standard out and standard err are directed to `System.out` and `System.err`.

- `public MysqldResource();`

The basedir is defaulted to a subdirectory of the `java.io.tmpdir`. Output for standard out and standard err are directed to `System.out` and `System.err`;

26.5.4.1.2. MysqldResource Methods

`MysqldResource` API includes the following methods:

- `void start(String threadName, Map mysqlArgs);`

Deploys and starts MySQL. The "threadName" string is used to name the thread which actually performs the execution of the

MySQL command line. The map is the set of arguments and their values to be passed to the command line.

- `void shutdown();`
Shuts down the MySQL instance managed by the `MysqldResource` object.
- `Map getServerOptions();`
Returns a map of all the options and their current (or default, if not running) options available for the MySQL database.
- `boolean isRunning();`
Returns true if the MySQL database is running.
- `boolean isReadyForConnections();`
Returns true once the database reports that is ready for connections.
- `void setKillDelay(int millis);`
The default “Kill Delay” is 30 seconds. This represents the amount of time to wait between the initial request to shutdown and issuing a “force kill” if the database has not shutdown by itself.
- `void addCompletionListenser(Runnable listener);`
Allows for applications to be notified when the server process completes. Each “listener” will be fired off in its own thread.
- `String getVersion();`
Returns the version of MySQL.
- `void setVersion(int MajorVersion, int minorVersion, int patchLevel);`
The standard distribution comes with only one version of MySQL packaged. However, it is possible to package multiple versions, and specify which version to use.

26.5.5. Connector/MXJ Notes and Tips

This section contains notes and tips on using the Connector/MXJ component within your applications.

26.5.5.1. Creating your own Connector/MXJ Package

If you want to create a custom Connector/MXJ package that includes a specific `mysqld` version or platform then you must extract and rebuild the `connector-mxj.jar` (Connector/MXJ v5.0.3 or earlier) or `connector-mxj-db-files.jar` (Connector/MXJ v5.0.4 or later) file.

First, you should create a new directory into which you can extract the current `connector-mxj.jar`:

```
shell> mkdir custom-mxj
shell> cd custom-mxj
shell> jar -xf connector-mxj.jar
shell> ls
5-0-22/
ConnectorMXJObjectTestExample.class
ConnectorMXJUrlTestExample.class
META-INF/
TestDb.class
com/
kill.exe
```

If you are using Connector/MXJ v5.0.4 or later, you should unpack the `connector-mxj-db-files.jar`:

```
shell> mkdir custom-mxj
shell> cd custom-mxj
shell> jar -xf connector-mxj-db-files.jar
shell> ls
5-0-27/
META-INF/
```

connector-mxj.properties

The MySQL version directory, [5-0-22](#) or [5-0-27](#) in the preceding examples, contains all of the files used to create an instance of MySQL when Connector/MXJ is executed. All of the files in this directory are required for each version of MySQL that you want to embed. Note as well the format of the version number, which uses hyphens instead of periods to separate the version number components.

Within the version specific directory are the platform specific directories, and archives of the [data](#) and [share](#) directory required by MySQL for the various platforms. For example, here is the listing for the default Connector/MXJ package:

```
shell>> ls
Linux-i386/
META-INF/
Mac_OS_X-ppc/
SunOS-sparc/
win-x86/
com/
data_dir.jar
share_dir.jar
win_share_dir.jar
```

Platform specific directories are listed by their OS and platform - for example the [mysqld](#) for Mac OS X PowerPC is located within the [Mac_OS_X-ppc](#) directory. You can delete directories from this location that you do not require, and add new directories for additional platforms that you want to support.

To add a platform specific [mysqld](#), create a new directory with the corresponding name for your operating system/platform. For example, you could add a directory for Mac OS X/Intel using the directory [Mac_OS_X-i386](#).

On Unix systems, you can determine the platform using [uname](#):

```
shell> uname -p
i386
```

Now you need to download or compile [mysqld](#) for the MySQL version and platform you want to include in your custom [connector-mxj.jar](#) package into the new directory.

Create a file called [version.txt](#) in the OS/platform directory you have just created that contains the version string/path of the [mysqld](#) binary. For example:

```
mysql-5.0.22-osx10.3-i386/bin/mysqld
```

You can now recreate the [connector-mxj.jar](#) file with the added [mysqld](#):

```
shell> cd custom-mxj
shell> jar -cf ../connector-mxj.jar *
```

For Connector/MXJ v5.0.4 and later, you should repackage to the [connector-mxj-db-files.jar](#):

```
shell> cd custom-mxj
shell> jar -cf ../connector-mxj-db-files.jar *
```

You should test this package using the steps outlined in [Section 26.5.2.3, “Connector/MXJ Quick Start Guide”](#).

Note

Because the [connector-mxj-db-files.jar](#) file is separate from the main Connector/MXJ classes you can distribute different [connector-mxj-db-files.jar](#) files to different hosts or for different projects without having to create a completely new main [connector-mxj.jar](#) file for each one.

26.5.5.2. Deploying Connector/MXJ with a pre-configured database

To include a pre-configured/populated database within your Connector/MXJ JAR file you must create a custom [data_dir.jar](#) file, as included within the main [connector-mxj.jar](#) (Connector/MXJ 5.0.3 or earlier) or [connector-mxj-db-files.jar](#) (Connector/MXJ 5.0.4 or later) file:

1. First extract the `connector-mxj.jar` file, as outlined in the previous section (see [Section 26.5.5.1, “Creating your own Connector/MXJ Package”](#)).
2. First, create your database and populate the database with the information you require in an existing instance of MySQL - including Connector/MXJ instances. Data file formats are compatible across platforms.
3. Shutdown the instance of MySQL.
4. Create a JAR file of the data directory and databases that you want to include your Connector/MXJ package. You should include the `mysql` database, which includes user authentication information, in addition to the specific databases you want to include. For example, to create a JAR of the `mysql` and `mxjtest` databases:

```
shell> jar -cf ../data_dir.jar mysql mxjtest
```

5. For Connector/MXJ 5.0.3 or earlier, copy the `data_dir.jar` file into the extracted `connector-mxj.jar` directory, and then create an archive for `connector-mxj.jar`.

For Connector/MXJ 5.0.4 or later, copy the `data_dir.jar` file into the extracted `connector-mxj-db-files.jar` directory, and then create an archive for `connector-mxj-db-files.jar`.

Note that if you are create databases using the InnoDB engine, you must include the `ibdata.*` and `ib_logfile*` files within the `data_dir.jar` archive.

26.5.5.3. Running within a JMX Agent (custom)

As a JMX MBean, MySQL Connector/MXJ requires a JMX v1.2 compliant MBean container, such as JBoss version 4. The MBean will uses the standard JMX management APIs to present (and allow the setting of) parameters which are appropriate for that platform.

If you are not using the SUN Reference implementation of the JMX libraries, you should skip this section. Or, if you are deploying to JBoss, you also may wish to skip to the next section.

We want to see the `MysqldDynamicMBean` in action inside of a JMX agent. In the `com.mysql.management.jmx.sunri` package is a custom JMX agent with two MBeans:

1. the `MysqldDynamicMBean`, and
2. a `com.sun.jdmk.comm.HtmlAdaptorServer`, which provides a web interface for manipulating the beans inside of a JMX agent.

When this very simple agent is started, it will allow a MySQL database to be started and stopped with a web browser.

1. Complete the testing of the platform as above.
 - current JDK, JUnit, Connector/J, MySQL Connector/MXJ
 - this section *requires* the SUN reference implementation of JMX
 - `PATH`, `JAVA_HOME`, `ANT_HOME`, `CLASSPATH`

2. If not building from source, skip to next step

rebuild with the "sunri.present"

```
ant -Dsunri.present=true dist
re-run tests:
java junit.textui.TestRunner com.mysql.management.AllTestsSuite
```

3. launch the test agent from the command line:

```
java com.mysql.management.jmx.sunri.MysqldTestAgentSunHtmlAdaptor &
```

4. from a browser:

```
http://localhost:9092/
```

5. under MysqlAgent,

```
select "name=mysqlld"
```

6. Observe the MBean View
7. scroll to the bottom of the screen press the STARTMYSQLD button
8. click [Back to MBean View](#)
9. scroll to the bottom of the screen press STOPMYSQLD button
10. kill the java process running the Test Agent (jmx server)

26.5.5.4. Deployment in a standard JMX Agent environment (JBoss)

Once there is confidence that the MBean will function on the platform, deploying the MBean inside of a standard JMX Agent is the next step. Included are instructions for deploying to JBoss.

1. Ensure a current version of java development kit (v1.4.x), see above.
 - Ensure [JAVA_HOME](#) is set (JBoss requires [JAVA_HOME](#))
 - Ensure [JAVA_HOME/bin](#) is in the [PATH](#) (You will NOT need to set your CLASSPATH, nor will you need any of the jars used in the previous tests).
2. Ensure a current version of JBoss (v4.0RC1 or better)

```
http://www.jboss.org/index.html
select "Downloads"
select "jboss-4.0.zip"
pick a mirror
unzip ~/dload/jboss-4.0.zip
create a JBOSS_HOME environment variable set to the unzipped directory
unix only:
cd $JBOSS_HOME/bin
chmod +x *.sh
```

3. Deploy (copy) the [connector-mxj.jar](#) to [\\$JBOSS_HOME/server/default/lib](#).
4. Deploy (copy) [mysql-connector-java-3.1.4-beta-bin.jar](#) to [\\$JBOSS_HOME/server/default/lib](#).
5. Create a [mxjtest.war](#) directory in [\\$JBOSS_HOME/server/default/deploy](#).
6. Deploy (copy) [index.jsp](#) to [\\$JBOSS_HOME/server/default/deploy/mxjtest.war](#).
7. Create a [mysqlld-service.xml](#) file in [\\$JBOSS_HOME/server/default/deploy](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <mbean code="com.mysql.management.jmx.jboss.JBossMysqlldDynamicMBean"
    name="mysql:type=service,name=mysqlld">
    <attribute name="datadir">/tmp/xxx_data_xxx</attribute>
    <attribute name="autostart">true</attribute>
  </mbean>
</server>
```

8. Start jboss:
 - on unix: [\\$JBOSS_HOME/bin/run.sh](#)

- on windows: `%JBoss_HOME%\bin\run.bat`

Be ready: JBoss sends a lot of output to the screen.

9. When JBoss seems to have stopped sending output to the screen, open a web browser to: <http://localhost:8080/jmx-console>
10. Scroll down to the bottom of the page in the `mysql` section, select the bulleted `mysql` link.
11. Observe the JMX MBean View page. MySQL should already be running.
12. (If "autostart=true" was set, you may skip this step.) Scroll to the bottom of the screen. You may press the INVOKE button to stop (or start) MySQL observe `Operation completed successfully without a return value`. Click `Back to MBean View`
13. To confirm MySQL is running, open a web browser to <http://localhost:8080/mxjtest/> and you should see that

```
SELECT 1
```


returned with a result of

```
1
```
14. Guided by the `$JBoss_HOME/server/default/deploy/mxjtest.war/index.jsp` you will be able to use MySQL in your Web Application. There is a `test` database and a `root` user (no password) ready to experiment with. Try creating a table, inserting some rows, and doing some selects.
15. Shut down MySQL. MySQL will be stopped automatically when JBoss is stopped, or: from the browser, scroll down to the bottom of the MBean View press the stop service INVOKE button to halt the service. Observe `Operation completed successfully without a return value`. Using `ps` or `task manager` see that MySQL is no longer running

As of 1.0.6-beta version is the ability to have the MBean start the MySQL database upon start up. Also, we've taken advantage of the JBoss life-cycle extension methods so that the database will gracefully shut down when JBoss is shutdown.

26.5.6. Connector/MXJ Support

There are a wide variety of options available for obtaining support for using Connector/MXJ. You should contact the Connector/MXJ community for help before reporting a potential bug or problem. See [Section 26.5.6.1, "Connector/MXJ Community Support"](#).

26.5.6.1. Connector/MXJ Community Support

MySQL AB provides assistance to the user community by means of a number of mailing lists and web based forums.

You can find help and support through the [MySQL and Java](#) mailing list.

For information about subscribing to MySQL mailing lists or to browse list archives, visit <http://lists.mysql.com/>. See [Section 1.7.1, "MySQL Mailing Lists"](#).

Community support from experienced users is also available through the [MyODBC Forum](#). You may also find help from other users in the other MySQL Forums, located at <http://forums.mysql.com>. See [Section 1.7.2, "MySQL Community Support at the MySQL Forums"](#).

26.5.6.2. How to Report Connector/MXJ Problems

If you encounter difficulties or problems with Connector/MXJ, contact the Connector/MXJ community [Section 26.5.6.1, "Connector/MXJ Community Support"](#).

If reporting a problem, you should ideally include the following information with the email:

- Operating system and version

- Connector/MXJ version
- MySQL server version
- Copies of error messages or other unexpected output
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

If you believe the problem to be a bug, then you must report the bug through <http://bugs.mysql.com/>.

26.5.6.3. Connector/MXJ Change History

The Connector/MXJ Change History (Changelog) is located with the main Changelog for MySQL. See [Section C.6, “MySQL Connector/MXJ Change History”](#).

26.6. Connector/PHP

The PHP distribution and documentation are available from the PHP Web site. MySQL provides the `mysql` and `mysqli` extensions for the Windows operating system on <http://dev.mysql.com/downloads/connector/php/> for MySQL version 4.1.16 and higher, MySQL 5.0.18, and MySQL 5.1. You can find information why you should preferably use the extensions provided by MySQL on that page. For platforms other than Windows, you should use the `mysql` or `mysqli` extensions shipped with the PHP sources. See [Section 25.3, “MySQL PHP API”](#).

Chapter 27. MySQL Proxy

The MySQL Proxy is an application that communicates over the network using the MySQL Network Protocol and provides communication between one or more MySQL servers and one or more MySQL clients. In the most basic configuration, MySQL Proxy simply passes on queries from the client to the MySQL Server and returns the responses from the MySQL Server to the client.

Because MySQL Proxy uses the MySQL network protocol, any MySQL compatible client (include the command line client, any clients using the MySQL client libraries, and any connector that supports the MySQL network protocol) can connect to the proxy without modification.

In addition to the basic pass-through configuration, the MySQL Proxy is also capable of monitoring and altering the communication between the client and the server. This interception of the queries enables you to add profiling. The interception of the exchanges is scriptable using the Lua scripting language.

By intercepting the queries from the client, the proxy can insert additional queries into the list of queries sent to the server, and remove the additional results when they are returned by the server. Using this functionality you can add informational statements to each query, for example to monitor their execution time or progress, and separately log the results, while still returning the results from the original query to the client.

The proxy allows you to perform additional monitoring, filtering or manipulation on queries without you having to make any modifications to the client and without the client even being aware that it is communicating with anything but a genuine MySQL server.

Warning

MySQL Proxy is currently an Alpha release and should not be used within production environments.

Important

MySQL Proxy is compatible with MySQL 5.0.x or later. Testing has not been performed with Version 4.1. Please provide feedback on your experiences via the [MySQL Proxy Forum](#).

27.1. MySQL Proxy Supported Platforms

MySQL Proxy is currently available as a pre-compiled binary for the following platforms:

- Linux (including RedHat, Fedora, Debian, SuSE) and derivatives.
- Mac OS X
- FreeBSD
- IBM AIX
- Sun Solaris

Other Unix/Linux platforms not listed should be compatible by using the source package and building MySQL Proxy locally.

System requirements for the MySQL Proxy application are the same as the main MySQL server. Currently MySQL Proxy is compatible only with MySQL 5.0.1 and later. MySQL Proxy is provided as a standalone, statically linked binary. You do not need to have MySQL or Lua installed.

27.2. Installing MySQL Proxy

You have three choices for installing MySQL Proxy:

- Pre-compiled binaries are available for a number of different platforms. See [Section 27.2.1, “Installing MySQL Proxy from a binary distribution”](#).
- You can install from the source code if you want to build on an environment not supported by the binary distributions. See [Section 27.2.2, “Installing MySQL Proxy from a source distribution”](#).

- The latest version of the MySQL proxy source code is available through a development repository is the best way to stay up to date with the latest fixes and revisions. See [Section 27.2.3, “Installing MySQL Proxy from the Subversion repository”](#).

27.2.1. Installing MySQL Proxy from a binary distribution

If you download the binary packages then you need only to extract the package and then copy the `mysql-proxy` file to your desired location. For example:

```
$ tar xzf mysql-proxy-0.5.0.tar.gz
$ cp ./mysql-proxy-0.5.0/sbin/mysql-proxy /usr/local/sbin
```

27.2.2. Installing MySQL Proxy from a source distribution

If you have downloaded the source package then you will need to compile the MySQL Proxy before using it. To build you will need to have the following installed:

- libevent 1.x or higher (1.3b or later is preferred)
- lua 5.1.x or higher
- glib2 2.6.0 or higher
- pkg-config
- MySQL 5.0.x or higher developer files

Once these components are installed, you need to configure and then build:

```
$ tar xzf mysql-proxy-0.5.0.tar.gz
$ cd mysql-proxy-0.5.0
$ ./configure
$ make
```

If you want to test the build, then use the `check` target to `make`:

```
$ make check
```

The tests try to connect to `localhost` using the `root` user. If you need to provide a password, set the `MYSQL_PASSWORD` environment variable:

```
$ MYSQL_PASSWORD=root_pwd make check
```

You can install using the `install` target:

```
$ make install
```

By default `mysql-proxy` is installed into `/usr/local/sbin/mysql-proxy`. The Lua example scripts are copied into `/usr/local/share`.

27.2.3. Installing MySQL Proxy from the Subversion repository

The MySQL Proxy source is available through a public Subversion repository and is the quickest way to get hold of the latest releases and fixes.

To build from the Subversion repository, you need the following components already installed:

- Subversion 1.3.0 or higher
- `libtool` 1.5 or higher

- [autoconf](#) 2.56 or higher
- [automake](#) 1.9 or higher
- [libevent](#) 1.x or higher (1.3b or later is preferred)
- [lua](#) 5.1.x or higher
- [glib2](#) 2.4.0 or higher
- [pkg-config](#)
- [MySQL](#) 5.0.x or higher developer files

To checkout a local copy of the Subversion repository, use [svn](#):

```
$ svn co http://svn.MySQL.com/svnpublic/mysql-proxy/ mysql-proxy
```

The above command will download a complete version of the Subversion repository for [mysql-proxy](#). The main source files are located within the [trunk](#) subdirectory. The configuration scripts need to be generated before you can configure and build [mysql-proxy](#). The [autogen.sh](#) script will generate the configuration scripts for you:

```
$ sh ./autogen.sh
```

The script creates the standard [configure](#) script, which you can then use to configure and build with [make](#):

```
$ ./configure
$ make
$ make install
```

If you want to create a standalone source distribution, identical to the source distribution available for download:

```
$ make distcheck
```

The above will create the file [mysql-proxy-0.5.0.tar.gz](#) within the current directory.

27.3. MySQL Proxy Command Line Options

To start [mysql-proxy](#) you can just run the command directly. However, for most situations you will want to specify at the very least the address/hostname and port number of the backend MySQL server to which the MySQL Proxy should pass on queries.

You can get a list of the supported command-line options using the [--help-all](#) command line option. The majority of these options set up the environment, either in terms of the address/port number that [mysql-proxy](#) should listen on for connections, or the onward connection to a MySQL server. A full description of the options is shown below:

- [--help-all](#) — show all help options.
- [--help-admin](#) — show options for the admin-module.
- [--help-proxy](#) — Show options for the proxy-module.
- [--admin-address=host:port](#) — specify the hostname (or IP address) and port for the administration port. The default is [localhost:4041](#).
- [--proxy-address=host:port](#) — the listening hostname (or IP address) and port of the proxy server. The default is [localhost:4040](#).
- [--proxy-read-only-address=host:port](#) — the listening hostname (or IP address) and port of the proxy server for read-only connections. The default is [localhost:4042](#).
- [--proxy-backend-addresses=host:port](#) — the hostname (or IP address) and port of the MySQL server to connect to. You can specify multiple backend servers by supplying multiple options. Clients are connected to each backend server in round-

robin fashion. For example, if you specify two servers A and B, the first client connection will go to server A; the second client connection to server B and the third client connection to server A.

- `--proxy-skip-profiling` — disables profiling of queries (tracking time statistics). The default is for tracking to be enabled.
- `--proxy-fix-bug-25371` — gets round an issue when connecting to a MySQL server later than 5.1.12 when using a MySQL client library of any earlier version.
- `--proxy-lua-script=file` — specify the Lua script file to be loaded. Note that the script file is not physically loaded and parsed until a connection is made. Also note that the specified Lua script is reloaded for each connection; if the content of the Lua script changes while `mysql-proxy` is running then the updated content will automatically be used when a new connection is made.
- `--daemon` — starts the proxy in daemon mode.
- `--pid-file=file` — sets the name of the file to be used to store the process ID.
- `--version` — show the version number.

The most common usage is as a simple proxy service (i.e. without addition scripting). For basic proxy operation you must specify at least one `proxy-backend-addresses` option to specify the MySQL server to connect to by default:

```
$ mysql-proxy
--proxy-backend-addresses=MySQL.example.com:3306
```

The default proxy port is `4040`, so you can connect to your MySQL server through the proxy by specifying the hostname and port details:

```
$ mysql --host=localhost --port=4040
```

If your server requires authentication information then this will be passed through natively without alteration by `mysql-proxy`, so you must also specify the authentication information if required:

```
$ mysql --host=localhost --port=4040 \
--user=username --password=password
```

You can also connect to a read-only port (which filters out `UPDATE` and `INSERT` queries) by connecting to the read-only port. By default the hostname is the default, and the port is `4042`, but you can alter the host/port information by using the `--proxy-read-only-address` command line option.

For more detailed information on how to use these command line options, and `mysql-proxy` in general in combination with Lua scripts, see [Section 27.5, “Using MySQL Proxy”](#).

27.4. MySQL Proxy Scripting

You can control how MySQL Proxy manipulates and works with the queries and results that are passed on to the MySQL server through the use of the embedded Lua scripting language. You can find out more about the Lua programming language from the [Lua Website](#).

The primary interaction between MySQL Proxy and the server is provided by defining one or more functions through an Lua script. A number of functions are supported, according to different events and operations in the communication sequence between a client and one or more backend MySQL servers:

- `connect_server()` — this function is called each time a connection is made to MySQL Proxy from a client. You can use this function during load-balancing to intercept the original connection and decide which server the client should ultimately be attached to. If you don't define a special solution, then a simple round-robin style distribution is used by default.
- `read_handshake()` — this function is called when the initial handshake information is returned by the server. You can capture the handshake information returned and provide additional checks before the authorization exchange takes place.
- `read_auth()` — this function is called when the authorization packet (username, password, default database) are submitted by the client to the server for authentication.

- `read_auth_result()` — this function is called when the server returns an authorization packet to the client indicating whether the authorization succeeded.
- `read_query()` — this function is called each time a query is sent by the client to the server. You can use this to edit and manipulate the original query, including adding new queries before and after the original statement. You can also use this function to return information directly to the client, bypassing the server, which can be useful to filter unwanted queries or queries that exceed known limits.
- `read_query_result()` — this function is called each time a result is returned from the server, providing you have manually injected queries into the query queue. If you have not explicitly inject queries within the `read_query()` function then this function is not triggered. You can use this to edit the result set, or to remove or filter the result sets generated from additional queries you injected into the queue when using `read_query()`.

The table below describes the direction of flow of information at the point when the function is triggered.

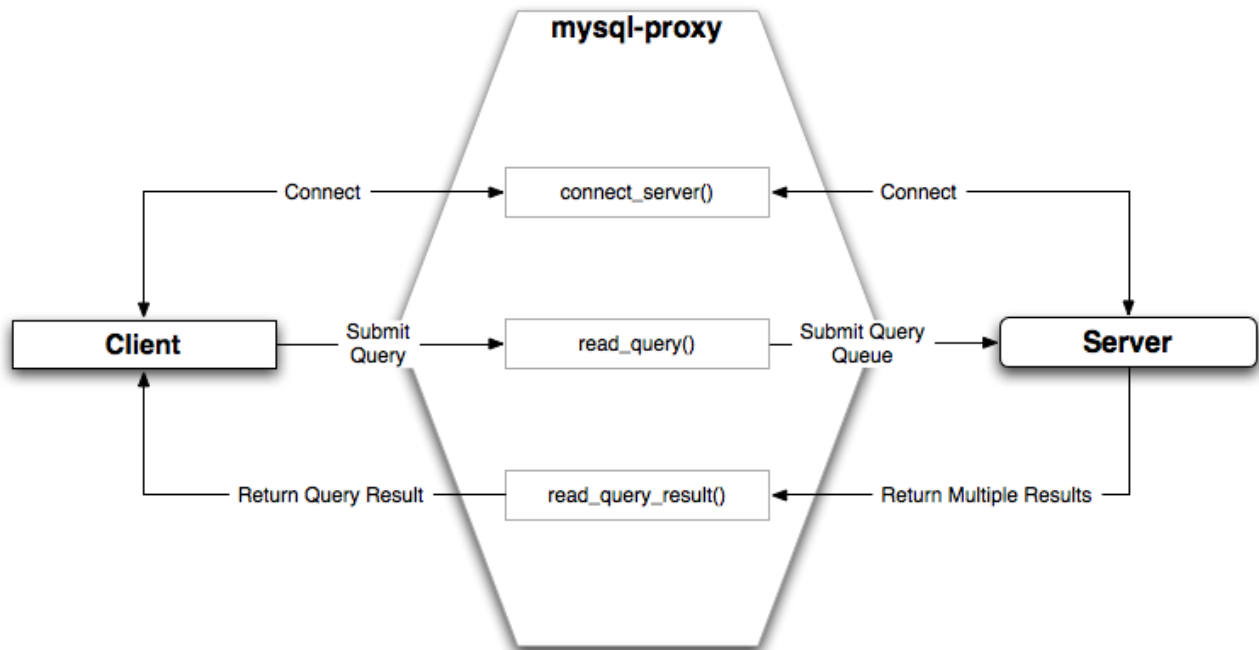
Function	Supplied Information	Direction
<code>connect_server()</code>	None	Client to Server
<code>read_handshake()</code>	Handshake packet	Server to Client
<code>read_auth()</code>	Authorization packet	Client to Server
<code>read_auth_result()</code>	Authorization response	Server to Client
<code>read_query()</code>	Query	Client to Server
<code>read_query_result()</code>	Query result	Server to Client

By default, all functions return a result that indicates that the data should be passed on to the client or server (depending on the direction of the information being transferred). This return value can be overridden by explicitly returning a constant indicating that a particular response should be sent. For example, it is possible to construct result set information by hand within `read_query()` and to return the resultset directly to the client without ever sending the original query to the server.

In addition to these functions, a number of built-in structures provide control over how MySQL Proxy forwards on queries and returns the results by providing a simplified interface to elements such as the list of queries and the groups of result sets that are returned.

27.4.1. Proxy Scripting Sequence During Query Injection

The figure below gives an example of how the proxy might be used when injecting queries into the query queue. Because the proxy sits between the client and MySQL server, what the proxy sends to the server, and the information that the proxy ultimately returns to the client do not have to match correlate. Once the client has connected to the proxy, the following sequence occurs for each individual query sent by the client.



1. The client submits one query to the proxy, the `read_query()` function within the proxy is triggered. The function adds the query to the query queue.
2. Once manipulation by `read_query()` has completed, the queries are submitted, sequentially, to the MySQL server.
3. The MySQL server returns the results from each query, one result set for each query submitted. The `read_query_result()` function is triggered for each result set, and each invocation can decide which result set to return to the client

For example, you can queue additional queries into the global query queue to be processed by the server. This can be used to add statistical information by adding queries before and after the original query, changing the original query:

```
SELECT * FROM City;
```

Into a sequence of queries:

```
SELECT NOW();
SELECT * FROM City;
SELECT NOW();
```

You can also modify the original statement, for example to add `EXPLAIN` to each statement executed to get information on how the statement was processed, again altering our original SQL statement into a number of statements:

```
SELECT * FROM City;
EXPLAIN SELECT * FROM City;
```

In both of these examples, the client would have received more result sets than expected. Regardless of how you manipulate the incoming query and the returned result, the number of queries returned by the proxy must match the number of original queries sent by the client.

You could adjust the client to handle the multiple result sets sent by the proxy, but in most cases you will want the existence of the proxy to remain transparent. To ensure that the number of queries and result sets match, you can use the MySQL Proxy `read_query_result()` to extract the additional result set information and return only the result set the client originally requested back to the client. You can achieve this by giving each query that you add to the query queue a unique ID, and then filter out queries that do not match the original query ID when processing them with `read_query_result()`.

27.4.2. Internal Structures

There are a number of internal structures within the scripting element of MySQL Proxy. The primary structure is `proxy` and this provides an interface to the many common structures used throughout the script, such as connection lists and configured backend servers. Other structures, such as the incoming packet from the client and result sets are only available within the context of one of the scriptable functions.

Attribute	Description
<code>connection</code>	A structure containing the active client connections. For a list of attributes, see <code>proxy.connection</code> .
<code>servers</code>	A structure containing the list of configured backend servers. For a list of attributes, see <code>proxy.servers</code> .
<code>queries</code>	A structure containing the queue of queries that will be sent to the server during a single client query. For a list of attributes, see <code>proxy.queries</code> .
<code>PROXY_VERSION</code>	The version number of MySQL Proxy, encoded in hex. You can use this to check that the version number supports a particular option from within the Lua script. Note that the value is encoded as a hex value, so to check the version is at least 0.5.1 you compare against <code>0x00501</code> .

`proxy.connection`

The `proxy.connection` object is read only, and provides information about the current connection.

Attribute	Description
<code>thread_id</code>	The thread ID of the connection.
<code>backend_ndx</code>	The ID of the server used for this connection. This is an ID valid against the list of configured servers available through the <code>proxy.servers</code> object.

`proxy.servers`

The `proxy.servers` table is partially writable and contains an array of all the configured backend servers and the server metadata (IP address, status, etc.). You can determine the array index of the current connection using `proxy.connection["backend_ndx"]` which is the index into this table of the backend server being used by the active connection.

The attributes for each entry within the `proxy.servers` table is shown in this table.

Attribute	Description
<code>address</code>	The hostname/port combination used for this connection
<code>connected_clients</code>	The number of clients currently connected.
<code>state</code>	The status of the backend server. See Section 27.4.2, “Internal Structures” [1694]

`proxy.queries`

The `proxy.queries` object is a queue representing the list of queries to be sent to the server. The queue is not populated automatically, but if you do not explicitly populate the queue then queries are passed on to the backend server verbatim. Also, if you do not populate the query queue by hand, then the `read_query_result()` function is not triggered.

The following methods are supported for populating the `proxy.queries` object.

Function	Description
<code>append(id, packet)</code>	Appends a query to the end of the query queue. The <code>id</code> is an integer identifier that you can use to recognize the query results when they are returned by the server. The packet should be a properly formatted query packet.
<code>prepend(id, packet)</code>	Prepends a query to the query queue. The <code>id</code> is an identifier that you can use to recognize the query results when they are returned by the server. The packet should be a properly formatted query packet.

Function	Description
<code>reset()</code>	Empties the query queue.
<code>len()</code>	Returns the number of query packets in the queue.

For example, you could append a query packet to the `proxy.queries` queue by using the `append()`:

```
proxy.queries:append(1, packet)
```

Proxy Return State Constants

The following constants are used internally by the proxy to specify the response to send to the client or server. All constants are exposed as values within the main `proxy` table.

Constant	Description
<code>PROXY_SEND_QUERY</code>	Causes the proxy to send the current contents of the queries queue to the server.
<code>PROXY_SEND_RESULT</code>	Causes the proxy to send a result set back to the client.
<code>PROXY_IGNORE_RESULT</code>	Causes the proxy to drop the result set (nothing is returned to the client).

As constants, these entities are available without qualification in the Lua scripts. For example, at the end of the `read_query_result()` you might return `PROXY_IGNORE_RESULT`:

```
return proxy.PROXY_IGNORE_RESULT
```

Packet State Constants

The following states describe the status of a network packet. These items are entries within the main `proxy` table.

Constant	Description
<code>MYSQLD_PACKET_OK</code>	The packet is OK.
<code>MYSQLD_PACKET_ERR</code>	The packet contains error information.
<code>MYSQLD_PACKET_RAW</code>	The packet contains raw data.

Backend State/Type Constants

The following constants are used either to define the status of the backend server (the MySQL server to which the proxy is connected) or the type of backend server. These items are entries within the main `proxy` table.

Constant	Description
<code>BACKEND_STATE_UNKNOWN</code>	The current status is unknown.
<code>BACKEND_STATE_UP</code>	The backend is known to be up (available).
<code>BACKEND_STATE_DOWN</code>	The backend is known to be down (unavailable).
<code>BACKEND_TYPE_UNKNOWN</code>	Backend type is unknown.
<code>BACKEND_TYPE_RW</code>	Backend is available for read/write.
<code>BACKEND_TYPE_RO</code>	Backend is available only for read-only use.

Server Command Constants

The following values are used in the packets exchanged between the client and server to identify the information in the rest of the packet. These items are entries within the main `proxy` table. The packet type is defined as the first character in the sent packet. For example, when intercepting packets from the client to edit or monitor a query you would check that the first byte of the packet was of type `proxy.COM_QUERY`.

Constant	Description
COM_SLEEP	Sleep
COM_QUIT	Quit
COM_INIT_DB	Initialize database
COM_QUERY	Query
COM_FIELD_LIST	Field List
COM_CREATE_DB	Create database
COM_DROP_DB	Drop database
COM_REFRESH	Refresh
COM_SHUTDOWN	Shutdown
COM_STATISTICS	Statistics
COM_PROCESS_INFO	Process List
COM_CONNECT	Connect
COM_PROCESS_KILL	Kill
COM_DEBUG	Debug
COM_PING	Ping
COM_TIME	Time
COM_DELAYED_INSERT	Delayed insert
COM_CHANGE_USER	Change user
COM_BINLOG_DUMP	Binlog dump
COM_TABLE_DUMP	Table dump
COM_CONNECT_OUT	Connect out
COM_REGISTER_SLAVE	Register slave
COM_STMT_PREPARE	Prepare server-side statement
COM_STMT_EXECUTE	Execute server-side statement
COM_STMT_SEND_LONG_DATA	Long data
COM_STMT_CLOSE	Close server-side statement
COM_STMT_RESET	Reset statement
COM_SET_OPTION	Set option
COM_STMT_FETCH	Fetch statement
COM_DAEMON	Daemon (MySQL 5.1 only)
COM_ERROR	Error

MySQL Type Constants

These constants are used to identify the field types in the query result data returned to clients from the result of a query. These items are entries within the main [proxy](#) table.

Constant	Field Type
MYSQL_TYPE_DECIMAL	Decimal
MYSQL_TYPE_NEWDECIMAL	Decimal (MySQL 5.0 or later)
MYSQL_TYPE_TINY	Tiny
MYSQL_TYPE_SHORT	Short
MYSQL_TYPE_LONG	Long
MYSQL_TYPE_FLOAT	Float
MYSQL_TYPE_DOUBLE	Double

Constant	Field Type
<code>MYSQL_TYPE_NULL</code>	Null
<code>MYSQL_TYPE_TIMESTAMP</code>	Timestamp
<code>MYSQL_TYPE_LONGLONG</code>	Long long
<code>MYSQL_TYPE_INT24</code>	Integer
<code>MYSQL_TYPE_DATE</code>	Date
<code>MYSQL_TYPE_TIME</code>	Time
<code>MYSQL_TYPE_DATETIME</code>	Datetime
<code>MYSQL_TYPE_YEAR</code>	Year
<code>MYSQL_TYPE_NEWDATE</code>	Date (MySQL 5.0 or later)
<code>MYSQL_TYPE_ENUM</code>	Enumeration
<code>MYSQL_TYPE_SET</code>	Set
<code>MYSQL_TYPE_TINY_BLOB</code>	Tiny Blob
<code>MYSQL_TYPE_MEDIUM_BLOB</code>	Medium Blob
<code>MYSQL_TYPE_LONG_BLOB</code>	Long Blob
<code>MYSQL_TYPE_BLOB</code>	Blob
<code>MYSQL_TYPE_VAR_STRING</code>	Varstring
<code>MYSQL_TYPE_STRING</code>	String
<code>MYSQL_TYPE_TINY</code>	Tiny (compatible with <code>MYSQL_TYPE_CHAR</code>)
<code>MYSQL_TYPE_ENUM</code>	Enumeration (compatible with <code>MYSQL_TYPE_INTERVAL</code>)
<code>MYSQL_TYPE_GEOMETRY</code>	Geometry
<code>MYSQL_TYPE_BIT</code>	Bit

27.4.3. Capturing a connection with `connect_server()`

When the proxy accepts a connection from a MySQL client, the `connect_server()` function is called.

There are no arguments to the function, but you can use and if necessary manipulate the information in the `proxy.connection` table, which is unique to each client session.

For example, if you have multiple backend servers then you can set the server to be used by that connection by setting the value of `proxy.connection.backend_ndx` to a valid server number. The code below will choose between two servers based on whether the current time in minutes is odd or even:

```
function connect_server()
    print("--> a client really wants to talk to a server")
    if (tonumber(os.date("%M")) % 2 == 0) then
        proxy.connection.backend_ndx = 2
        print("Choosing backend 2")
    else
        proxy.connection.backend_ndx = 1
        print("Choosing backend 1")
    end
    print("Using " .. proxy.servers[proxy.connection.backend_ndx].address)
end
```

In this example the IP address/port combination is also displayed by accessing the information from the internal `proxy.servers` table.

27.4.4. Examining the handshake with `read_handshake()`

Handshake information is sent by the server to the client after the initial connection (through `connect_server()`) has been made. The handshake information contains details about the MySQL version, the ID of the thread that will handle the connection information, and the IP address of the client and server. This information is exposed through a Lua table as the only argument to the function.

- `mysqlnd_version` — the version of the MySQL server.
- `thread_id` — the thread ID.
- `scramble` — the password scramble buffer.
- `server_addr` — the IP address of the server.
- `client_addr` — the IP address of the client.

For example, you can print out the handshake data and refuse clients by IP address with the following function:

```
function read_handshake( auth )
    print("<-- let's send him some information about us")
    print("    mysqlnd-version: " .. auth.mysqlnd_version)
    print("    thread-id       : " .. auth.thread_id)
    print("    scramble-buf    : " .. string.format("%q", auth.scramble))
    print("    server-addr     : " .. auth.server_addr)
    print("    client-addr      : " .. auth.client_addr)

    if not auth.client_addr:match("^127.0.0.1:") then
        proxy.response.type = proxy.MYSQLD_PACKET_ERR
        proxy.response.errmsg = "only local connects are allowed"

        print("we don't like this client");

        return proxy.PROXY_SEND_RESULT
    end
end
```

Note that you have to return an error packet to the client by using `proxy.PROXY_SEND_RESULT`.

27.4.5. Examining the authentication credentials with `read_auth()`

The `read_auth()` function is triggered when an authentication handshake is initiated by the client. In the execution sequence, `read_auth()` occurs immediately after `read_handshake()`, so the server selection has already been made, but the connection and authorization information has not yet been provided to the backend server.

The function accepts a single argument, an Lua table containing the authorization information for the handshake process. The entries in the table are:

- `username` — the user login for connecting to the server.
- `password` — the password, encrypted, to be used when connecting.
- `default_db` — the default database to be used once the connection has been made.

For example, you can print the username and password supplied during authorization using:

```
function read_auth( auth )
    print("    username      : " .. auth.username)
    print("    password      : " .. string.format("%q", auth.password))
end
```

You can interrupt the authentication process within this function and return an error packet back to the client by constructing a new packet and returning `proxy.PROXY_SEND_RESULT`:

```
proxy.response.type = proxy.MYSQLD_PACKET_ERR
proxy.response.errmsg = "Logins are not allowed"
return proxy.PROXY_SEND_RESULT
```

27.4.6. Accessing authentication information with `read_auth_result()`

The return packet from the server during authentication is captured by `read_auth_result()`. The only argument to this function is the authentication packet returned by the server. As the packet is a raw MySQL network protocol packet, you must access the first byte to identify the packet type and contents. The `MYSQLD_PACKET_ERR` and `MYSQLD_PACKET_OK` constants can be used to identify

whether the authentication was successful:

```
function read_auth_result( auth )
    local state = auth.packet:byte()

    if state == proxy.MYSQLD_PACKET_OK then
        print("<-- auth ok");
    elseif state == proxy.MYSQLD_PACKET_ERR then
        print("<-- auth failed");
    else
        print("<-- auth ... don't know: " .. string.format("%q", auth.packet));
    end
end
```

27.4.7. Manipulating Queries with `read_query()`

The `read_query()` function is called once for each query submitted by the client and accepts a single argument, the query packet that was provided. To access the content of the packet you must parse the packet contents manually.

For example, you can intercept a query packet and print out the contents using the following function definition:

```
function read_query( packet )
    if packet:byte() == proxy.COM_QUERY then
        print("we got a normal query: " .. packet:sub(2))
    end
end
```

This example checks the first byte of the packet to determine the type. If the type is `COM_QUERY` (see [Section 27.4.2, “Internal Structures” \[1694\]](#)), then we extract the query from the packet and print it out. The structure of the packet type supplied is important. In the case of a `COM_QUERY` packet, the remaining contents of the packet are the text of the query string. In this example, no changes have been made to the query or the list of queries that will ultimately be sent to the MySQL server.

To modify a query, or add new queries, you must populate the query queue (`proxy.queries`) and then execute the queries that you have placed into the queue. If you do not modify the original query or the queue, then the query received from the client is sent to the MySQL server verbatim.

When adding queries to the queue, you should follow these guidelines:

- The packets inserted into the queue must be valid query packets. For each packet, you must set the initial byte to the packet type. If you are appending a query, you can append the query statement to the rest of the packet.
- Once you add a query to the queue, the queue is used as the source for queries sent to the server. If you add a query to the queue to add more information, you must also add the original query to the queue or it will not be executed.
- Once the queue has been populated, you must set the return value from `read_query()` to indicate whether the query queue should be sent to the server.
- When you add queries to the queue, you should add an ID. The ID you specify is returned with the result set so that you identify each query and corresponding result set. The ID has no other purpose than as an identifier for correlating the query and resultset. When operating in a passive mode, during profiling for example, you want to identify the original query and the corresponding resultset so that the results expect by the client can be returned correctly.
- Unless your client is designed to cope with more result sets than queries, you should ensure that the number of queries from the client match the number of results sets returned to the client. Using the unique ID and removing result sets you inserted will help.

Normally, the `read_query()` and `read_query_result()` function are used in conjunction with each other to inject additional queries and remove the additional result sets. However, `read_query_result()` is only called if you populate the query queue within `read_query()`.

27.4.8. Manipulating Results with `read_query_result()`

The `read_query_result()` is called for each result set returned by the server only if you have manually injected queries into the query queue. If you have not manipulated the query queue then this function is not called. The function supports a single argument, the result packet, which provides a number of properties:

- `id` — the ID of the result set, which corresponds to the ID that was set when the query packet was submitted to the server when us-

ing `append(id)` on the query queue.

- `query` — the text of the original query.
- `query_time` — the number of microseconds required to receive the first row of a result set.
- `response_time` — the number of microseconds required to receive the last row of the result set.
- `resultset` — the content of the result set data.

By accessing the result information from the MySQL server you can extract the results that match the queries that you injected, return different result sets (for example, from a modified query), and even create your own result sets.

The Lua script below, for example, will output the query, followed by the query time and response time (i.e. the time to execute the query and the time to return the data for the query) for each query sent to the server:

```
function read_query( packet )
    if packet:byte() == proxy.COM_QUERY then
        print("we got a normal query: " .. packet:sub(2))

        proxy.queries:append(1, packet )

        return proxy.PROXY_SEND_QUERY
    end
end

function read_query_result(inj)
    print("query-time: " .. (inj.query_time / 1000) .. "ms")
    print("response-time: " .. (inj.response_time / 1000) .. "ms")
end
```

You can access the rows of returned results from the resultset by accessing the rows property of the resultset property of the result that is exposed through `read_query_result()`. For example, you can iterate over the results showing the first column from each row using this Lua fragment:

```
for row in inj.resultset.rows do
    print("injected query returned: " .. row[0])
end
```

Just like `read_query()`, `read_query_result()` can return different values for each result according to the result returned. If you have injected additional queries into the query queue, for example, then you will want to remove the results returned from those additional queries and only return the results from the query originally submitted by the client.

The example below injects additional `SELECT NOW()` statements into the query queue, given them a different ID to the ID of the original query. Within `read_query_result()`, if the ID for the injected queries is identified, we display the result row, and return the `proxy.PROXY_IGNORE_RESULT` from the function so that the result is not returned to the client. If the result is from any other query, we print out the query time information for the query and return the default, which passes on the result set unchanged. We could also have explicitly returned `proxy.PROXY_IGNORE_RESULT` to the MySQL client.

```
function read_query( packet )
    if packet:byte() == proxy.COM_QUERY then
        proxy.queries:append(2, string.char(proxy.COM_QUERY) .. "SELECT NOW()" )
        proxy.queries:append(1, packet )
        proxy.queries:append(2, string.char(proxy.COM_QUERY) .. "SELECT NOW()" )

        return proxy.PROXY_SEND_QUERY
    end
end

function read_query_result(inj)
    if inj.id == 2 then
        for row in inj.resultset.rows do
            print("injected query returned: " .. row[0])
        end
        return proxy.PROXY_IGNORE_RESULT
    else
        print("query-time: " .. (inj.query_time / 1000) .. "ms")
        print("response-time: " .. (inj.response_time / 1000) .. "ms")
    end
end
```

For further examples, see [Section 27.5, “Using MySQL Proxy”](#).

27.5. Using MySQL Proxy

There are a number of different ways to use MySQL Proxy. At the most basic level, you can allow MySQL Proxy to pass on queries from clients to a single server. To use MySQL proxy in this mode, you just have to specify the backend server that the proxy should connect to on the command line:

```
$ mysql-proxy --proxy-backend-addresses=sakila:3306
```

If you specify multiple backend MySQL servers then the proxy will connect each client to each server in a round-robin fashion. For example, imagine you have two MySQL servers, A and B. The first client to connect will be connected to server A, the second to server B, the third to server C. For example:

```
$ mysql-proxy \
  --proxy-backend-addresses=narcissus:3306 \
  --proxy-backend-addresses=nostromo:3306
```

When you have specified multiple servers in this way, the proxy will automatically identify when a MySQL server has become unavailable and mark it accordingly. New connections will automatically be attached to a server that is available, and a warning will be reported to the standard output from [mysql-proxy](#):

```
network-mysqld.c.367: connect(nostromo:3306) failed: Connection refused
network-mysqld-proxy.c.2405: connecting to backend (nostromo:3306) failed, marking it as down for ...
```

Lua scripts enable a finer level of control, both over the connections and their distribution and how queries and result sets are processed. When using an Lua script, you must specify the name of the script on the command line using the [--proxy-lua-script](#) option:

```
$ mysql-proxy --proxy-lua-script=mc.lua --proxy-backend-addresses=sakila:3306
```

When you specify a script, the script is not executed until a connection is made. This means that faults with the script will not be raised until the script is executed. Script faults will not affect the distribution of queries to backend MySQL servers.

Note

Because the script is not read until the connection is made, you can modify the contents of the Lua script file while the proxy is still running and the script will automatically be used for the next connection. This ensures that MySQL Proxy remains available because it does not have to be restarted for the changes to take effect.

27.5.1. Using the Administration Interface

The [mysql-proxy](#) administration interface can be accessed using any MySQL client using the standard protocols. You can use the administration interface to gain information about the proxy server as a whole - standard connections to the proxy are isolated to operate as if you were connected directly to the backend MySQL server. Currently, the interface supports a limited set of functionality designed to provide connection and configuration information.

Because connectivity is provided over the standard MySQL protocol, you must access this information using SQL syntax. By default, the administration port is configured as 4041. You can change this port number using the [--admin-address](#) command line option.

To get a list of the currently active connections to the proxy:

```
mysql> select * from proxy_connections;
+----+-----+-----+-----+
| id | type  | state | db   |
+----+-----+-----+-----+
| 0  | server | 0     |      |
| 1  | proxy  | 0     |      |
| 2  | server | 10    |      |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

To get the current configuration:

```
mysql> select * from proxy_config;
+-----+-----+
| option | value |
+-----+-----+
| admin.address | :4041 |
| proxy.address | :4040 |
| proxy.lua_script | mc.lua |
+-----+-----+
```

proxy.backend_addresses[0]	mysql:3306	
proxy.fix_bug_25371	0	
proxy.profiling	1	
+-----+-----+		

6 rows in set (0.01 sec)

Chapter 28. Extending MySQL

28.1. MySQL Internals

This chapter describes a lot of things that you need to know when working on the MySQL code. If you plan to contribute to MySQL development, want to have access to the bleeding-edge versions of the code, or just want to keep track of development, follow the instructions in [Section 2.9.3, “Installing from the Development Source Tree”](#). If you are interested in MySQL internals, you should also subscribe to our [internals](#) mailing list. This list has relatively low traffic. For details on how to subscribe, please see [Section 1.7.1, “MySQL Mailing Lists”](#). All developers at MySQL AB are on the [internals](#) list and we help other people who are working on the MySQL code. Feel free to use this list both to ask questions about the code and to send patches that you would like to contribute to the MySQL project!

28.1.1. MySQL Threads

The MySQL server creates the following threads:

- One thread manages TCP/IP file connection requests and creates a new dedicated thread to handle the authentication and SQL statement processing for each connection. (On Unix, this thread also manages Unix socket file connection requests.) On Windows, a similar thread manages shared-memory connection requests, and a thread manages named-pipe connection requests. Every client connection has its own thread, although the manager threads try to avoid creating threads by consulting the thread cache first to see whether a cached thread can be used for a new connection.
- On a master replication server, slave server connections are like client connections: There is one thread per connected slave.
- On a slave replication server, an I/O thread is started to connect to the master server and read updates from it. An SQL thread is started to apply updates read from the master. These two threads run independently and can be started and stopped independently.
- The signal thread handles all signals. This thread also normally handles alarms and calls `process_alarm()` to force timeouts on connections that have been idle too long.
- If **InnoDB** is used, there will be 4 additional threads by default. Those are file I/O threads, controlled by the `innodb_file_io_threads` parameter. See [Section 13.5.4, “InnoDB Startup Options and System Variables”](#).
- If `mysqld` is compiled with `-DUSE_ALARM_THREAD`, a dedicated thread that handles alarms is created. This is only used on some systems where there are problems with `sigwait()` or if you want to use the `thr_alarm()` code in your application without a dedicated signal handling thread.
- If the server is started with the `--flush_time=val` option, a dedicated thread is created to flush all tables every `val` seconds.
- Each table for which **INSERT DELAYED** statements are issued gets its own thread.
- If the event scheduler is active, there is one thread for the scheduler, and a thread for each event currently running.

`mysqladmin processlist` only shows the connection, **INSERT DELAYED**, replication threads, and event threads.

MySQL Enterprise

For expert advice on thread management subscribe to the MySQL Enterprise Monitor. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

28.1.2. MySQL Test Suite

The test system that is included in Unix source and binary distributions makes it possible for users and developers to perform regression tests on the MySQL code. These tests can be run on Unix.

The current set of test cases doesn't test everything in MySQL, but it should catch most obvious bugs in the SQL processing code, operating system or library issues, and is quite thorough in testing replication. Our goal is to have the tests cover 100% of the code. We welcome contributions to our test suite. You may especially want to contribute tests that examine the functionality critical to your system because this ensures that all future MySQL releases work well with your applications.

The test system consists of a test language interpreter (`mysqltest`), a Perl script to run all tests (`mysql-test-run.pl`), the actual test cases written in a special test language, and their expected results. To run the test suite on your system after a build, type `make`

`test` from the source root directory, or change location to the `mysql-test` directory and type `./mysql-test-run.pl`. If you have installed a binary distribution, change location to the `mysql-test` directory under the installation root directory (for example, `/usr/local/mysql/mysql-test`), and run `./mysql-test-run.pl`. All tests should succeed. If any do not, feel free to try to find out why and report the problem if it indicates a bug in MySQL. See [Section 1.8, “How to Report Bugs or Problems”](#).

If one test fails, you should run `mysql-test-run.pl` with the `--force` option to check whether any other tests fail.

If you have a copy of `mysqld` running on the machine where you want to run the test suite, you do not have to stop it, as long as it is not using ports `9306` or `9307`. If either of those ports is taken, you should set the `MTR_BUILD_THREAD` environment variable to an appropriate value, and the test suite will use a different set of ports for master, slave, NDB, and Instance Manager). For example:

```
shell> export MTR_BUILD_THREAD=31
shell> ./mysql-test-run.pl [options] [test_name]
```

In the `mysql-test` directory, you can run an individual test case with `./mysql-test-run.pl test_name`.

You can use the `mysqltest` language to write your own test cases. This is documented in the MySQL Test Framework manual, available at <http://dev.mysql.com/doc/>.

If you have a question about the test suite, or have a test case to contribute, send an email message to the MySQL [internals](#) mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#). This list does not accept attachments, so you should FTP all the relevant files to: <ftp://ftp.mysql.com/pub/mysql/upload/>

28.2. The MySQL Plugin Interface

MySQL 5.1 and up supports a plugin API that allows the loading and unloading of server components at runtime, without restarting the server. Currently, the plugin API supports creation of full-text parser plugins. Such a plugin can be used to replace or augment the built-in full-text parser. For example, a plugin can parse text into words using rules that differ from those used by the built-in parser. This can be useful if you need to parse text with characteristics different from those expected by the built-in parser.

The plugin interface is intended as the successor to the older user-defined function (UDF) interface. The plugin interface eventually will include an API for creating UDFs, and it is intended this plugin UDF API will replace the older non-plugin UDF API. After that point, it will be possible for UDFs to be revised for use as plugin UDFs so that they can take advantage of the better security and versioning capabilities of the plugin API. Eventually, support for the older UDF API will be phased out.

The plugin interface requires the `plugin` table in the `mysql` database. This table is created as part of the MySQL installation process. If you are upgrading from an older version to MySQL 5.1, you should run the `mysql_upgrade` command to create this table. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

28.2.1. Characteristics of the Plugin Interface

In some respects, the plugin API is similar to the older user-defined function (UDF) API that it supersedes, but the plugin API has several advantages over the older interface:

- The plugin framework is extendable to accommodate different kinds of plugins.

Some aspects of the plugin API are common to all types of plugins, but the API also allows for type-specific interface elements so that different types of plugins can be created. A plugin with one purpose can have an interface most appropriate to its own requirements and not the requirements of some other plugin type.

Although only the interface for full-text parser plugins is implemented currently, others can be added, such as an interface for UDF plugins.

- The plugin API includes versioning information.

The version information included in the plugin API enables a plugin library and each plugin that it contains to be self-identifying with respect to the API version that was used to build the library. If the API changes over time, the version numbers will change, but a server can examine a given plugin library's version information to determine whether it supports the plugins in the library.

There are two types of version numbers. The first is the version for the general plugin framework itself. Each plugin library includes this kind of version number. The second type of version applies to individual plugins. Each specific type of plugin has a version for its interface, so each plugin in a library has a type-specific version number. For example, library containing a full-text parsing plugin has a general plugin API version number, and the plugin has a version number specific to the full-text plugin interface.

- Plugin security is improved relative to the UDF interface.

The older interface for writing non-plugin UDFs allowed libraries to be loaded from any directory searched by the system's dynamic linker, and the symbols that identified the UDF library were relatively non-specific. The newer rules are more strict. A plugin library must be installed in a specific dedicated directory for which the location is controlled by the server and cannot be changed at runtime. Also, the library must contain specific symbols that identify it as a plugin library. The server will not load something as a plugin if it was not built as a plugin.

The newer plugin interface eliminates the security issues of the older UDF interface. When a UDF plugin type is implemented, that will allow non-plugin UDFs to be brought into the plugin framework and the older interface to be phased out.

The plugin implementation includes the following components:

Source files (the locations given indicate where the files are found in a MySQL source distribution):

- `include/mysql/plugin.h` exposes the public plugin API. This file should be examined by anyone who wants to write a plugin library.
- `sql/sql_plugin.h` and `sql/sql_plugin.cc` comprise the internal plugin implementation. These files need not be consulted by plugin writers. They may be of interest for those who want to know more about how the server handles plugins.

System table:

- The `plugin` table in the `mysql` database lists each installed plugin and is required for plugin use. For new MySQL installations, this table is created during the installation process. If you are upgrading from a version older than MySQL 5.1, you should run `mysql_upgrade` to update your system tables and create the `plugin` table (see [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#)).

SQL statements:

- `INSTALL PLUGIN` registers a plugin in the `plugin` table and loads the plugin code.
- `UNINSTALL PLUGIN` unregisters a plugin from the `plugin` table and unloads the plugin code.
- The `WITH PARSER` clause for full-text index creation associates a full-text parser plugin with a given `FULLTEXT` index.
- `SHOW PLUGINS` displays information about known plugins. The `PLUGINS` table in `INFORMATION_SCHEMA` also contains plugin information.

System variable:

- `plugin_dir` indicates the location of the directory where all plugins must be installed. The value of this variable can be specified at server startup with a `--plugin_dir=path` option.

28.2.2. Full-Text Parser Plugins

MySQL has a built-in parser that it uses by default for full-text operations (parsing text to be indexed, or parsing a query string to determine the terms to be used for a search). For full-text processing, “parsing” means extracting words from text or a query string based on rules that define which character sequences make up a word and where word boundaries lie.

When parsing for indexing purposes, the parser passes each word to the server, which adds it to a full-text index. When parsing a query string, the parser passes each word to the server, which accumulates the words for use in a search.

The parsing properties of the built-in full-text parser are described in [Section 11.8, “Full-Text Search Functions”](#). These properties include rules for determining how to extract words from text. The parser is influenced by certain system variables such as `ft_min_word_len` and `ft_max_word_len` that cause words shorter or longer to be excluded, and by the stopword list that identifies common words to be ignored.

The plugin API enables you to provide a full-text parser of your own so that you have control over the basic duties of a parser. A parser plugin can operate in either of two roles:

- The plugin can replace the built-in parser. In this role, the plugin reads the input to be parsed, splits it up into words, and passes the words to the server (either for indexing or for word accumulation).

One reason to use a parser this way is that you need to use different rules from those of the built-in parser for determining how to split up input into words. For example, the built-in parser considers the text “case-sensitive” to consist of two words “case” and “sensitive,” whereas an application might need to treat the text as a single word.

- The plugin can act in conjunction with the built-in parser by serving as a front end for it. In this role, the plugin extracts text from the input and passes the text to the parser, which splits up the text into words using its normal parsing rules. In particular, this parsing will be affected by the `ft_XXX` system variables and the stopwords list.

One reason to use a parser this way is that you need to index content such as PDF documents, XML documents, or `.doc` files. The built-in parser is not intended for those types of input but a plugin can pull out the text from these input sources and pass it to the built-in parser.

It is also possible for a parser plugin to operate in both roles. That is, it could extract text from non-plaintext input (the front end role), and also parse the text into words (thus replacing the built-in parser).

A full-text plugin is associated with full-text indexes on a per-index basis. That is, when you install a parser plugin initially, that does not cause it to be used for any full-text operations. It simply becomes available. For example, a full-text parser plugin becomes available to be named in a `WITH PARSER` clause when creating individual `FULLTEXT` indexes. To create such an index at table-creation time, do this:

```
CREATE TABLE t
(
  doc CHAR(255),
  FULLTEXT INDEX (doc) WITH PARSER my_parser
);
```

Or you can add the index after the table has been created:

```
ALTER TABLE t ADD FULLTEXT INDEX (doc) WITH PARSER my_parser;
```

The only SQL change for associating the parser with the index is the `WITH PARSER` clause. Searches are specified as before, with no changes needed for queries.

When you associate a parser plugin with a `FULLTEXT` index, the plugin is required for using the index. If the parser plugin is dropped, any index associated with it becomes unusable. Any attempt to use it a table for which a plugin is not available results in an error, although `DROP TABLE` is still possible.

28.2.3. INSTALL PLUGIN Syntax

```
INSTALL PLUGIN plugin_name SONAME 'plugin_library'
```

This statement installs a plugin.

plugin_name is the name of the plugin as defined in the plugin declaration structure contained in the library file. Plugin name case sensitivity is determined by the host system filename semantics.

plugin_library is the name of the shared library that contains the plugin code. The name includes the filename extension (for example, `libmyplugin.so` or `libmyplugin.dylib`).

The shared library must be located in the plugin directory (that is, the directory named by the `plugin_dir` system variable). The library must be in the plugin directory itself, not in a subdirectory. By default, `plugin_dir` is the directory named by the `pkglibdir` configuration variable, but it can be changed by setting the value of `plugin_dir` at server startup. For example, set its value in a `my.cnf` file:

```
[mysqld]
plugin_dir=/path/to/plugin/directory
```

If the value of `plugin_dir` is a relative pathname, it is taken to be relative to the MySQL base directory (the value of the `basedir` system variable).

INSTALL PLUGIN adds a line to the `mysql.plugin` table that describes the `plugin`. This table contains the plugin name and library filename.

INSTALL PLUGIN also loads and initializes the plugin code to make the plugin available for use. A plugin is initialized by executing its initialization function, which handles any setup that the plugin must perform before it can be used.

To use **INSTALL PLUGIN**, you must have the **INSERT privilege** for the `mysql.plugin` table.

At server startup, the server loads and initializes any plugin that is listed in the `mysql.plugin` table. This means that a plugin is installed with **INSTALL PLUGIN** only once, not every time the server starts. Plugin loading at startup does not occur if the server is started with the `--skip-grant-tables` option.

When the server shuts down, it executes the deinitialization function for each plugin that is loaded so that the plugin has a chance to perform any final cleanup.

To remove a plugin entirely, use the **UNINSTALL PLUGIN** statement:

To see what plugins are installed, use the **SHOW PLUGIN** statement.

If you recompile a plugin library and need to reinstall it, you can use either of the following procedures:

- Use **UNINSTALL PLUGIN** to uninstall all plugins in the library, install the new plugin library file in the plugin directory, and then use **INSTALL PLUGIN** to install all plugins in the library. This procedure has the advantage that it can be used without stopping the server. However, if the plugin library contains many plugins, you must issue many **INSTALL PLUGIN** and **UNINSTALL PLUGIN** statements.
- Alternatively, stop the server, install the new plugin library file in the plugin directory, and then restart the server.

28.2.4. UNINSTALL PLUGIN Syntax

```
UNINSTALL PLUGIN plugin_name
```

This statement removes an installed plugin. You cannot uninstall a plugin if any table that uses it is open.

plugin_name must be the name of some plugin that is listed in the `mysql.plugin` table. The server executes the plugin's deinitialization function and removes the row for the plugin from the `mysql.plugin` table, so that subsequent server restarts will not load and initialize the plugin. **UNINSTALL PLUGIN** does not remove the plugin's shared library file.

To use **UNINSTALL PLUGIN**, you must have the **DELETE** privilege for the `mysql.plugin` table.

Plugin removal has implications for the use of associated tables. For example, if a full-text parser plugin is associated with a **FULL-TEXT** index on the table, uninstalling the plugin makes the table unusable. Any attempt to access the table results in an error. The table cannot even be opened, so you cannot drop an index for which the plugin is used. This means that uninstalling a plugin is something to do with care unless you do not care about the table contents. If you are uninstalling a plugin with no intention of reinstalling it later and you care about the table contents, you should dump the table with `mysqldump` and remove the **WITH PARSE** clause from the dumped **CREATE TABLE** statement so that you can reload the table later. If you do not care about the table, **DROP TABLE** can be used even if any plugins associated with the table are missing.

28.2.5. Writing Plugins

This section describes the general and type-specific parts of the plugin API. It also provides a step-by-step guide to creating a plugin library. For example plugin source code, see the `plugin/fulltext` directory of a MySQL source distribution.

You can write plugins in C or C++ (or another language that can use C calling conventions). Plugins are loaded and unloaded dynamically, so your operating system must support dynamic loading and you must have compiled `mysqld` dynamically (not statically).

A plugin contains code that becomes part of the running server, so when you write a plugin, you are bound by any and all constraints that otherwise apply to writing server code. For example, you may have problems if you attempt to use functions from the `libstdc++` library. These constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to plu-

gins that were originally written for older servers. For information about these constraints, see [Section 2.9.2, “Typical configure Options”](#), and [Section 2.9.4, “Dealing with Problems Compiling MySQL”](#).

28.2.5.1. General Plugin Structures and Functions

Every plugin must have a general plugin declaration. The declaration corresponds to the `st_mysql_plugin` structure in the `plugin.h` file:

```
struct st_mysql_plugin
{
  int type;           /* the plugin type (a MYSQL_XXX_PLUGIN value) */
  void *info;         /* pointer to type-specific plugin descriptor */
  const char *name;   /* plugin name */
  const char *author; /* plugin author (for SHOW PLUGINS) */
  const char *descr;  /* general descriptive text (for SHOW PLUGINS) */
  int license;        /* the plugin license (PLUGIN_LICENSE_XXX) */
  int (*init)(void *); /* the function to invoke when plugin is loaded */
  int (*deinit)(void *); /* the function to invoke when plugin is unloaded */
  unsigned int version; /* plugin version (for SHOW PLUGINS) */
  struct st_mysql_show_var *status_vars;
  void * __reserved1; /* placeholder for system variables */
  void * __reserved2; /* placeholder for config options */
};
```

The `st_mysql_plugin` structure is common to every type of plugin. Its members should be filled in as follows:

- type**

The plugin type. This must be one of the plugin-type values from `plugin.h`. For a full-text parser plugin, the `type` value is `MYSQL_FTPARSER_PLUGIN`.

- info**

A pointer to the descriptor for the plugin. Unlike the general plugin declaration structure, this descriptor's structure depends on the particular type of plugin. Each descriptor has a version number that indicates the API version for that type of plugin, plus any other members needed. The descriptor for full-text plugins is described in [Section 28.2.5.2, “Type-Specific Plugin Structures and Functions”](#).

- name**

The plugin name. This is the name that will be listed in the `plugin` table and by which you refer to the plugin in SQL statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`.

- author**

The plugin author. This can be whatever you like.

- descr**

A general description of the plugin. This can be whatever you like.

- license**

The plugin license type. The value can be one of `PLUGIN_LICENSE_PROPRIETARY`, `PLUGIN_LICENSE_GPL`, or `PLUGIN_LICENSE_BSD`.

- init**

A once-only initialization function. This is executed when the plugin is loaded, which happens for `INSTALL PLUGIN` or, for plugins listed in the `plugin` table, at server startup. The function takes no arguments. It returns zero for success and non-zero for failure. If an `init` function is unneeded for a plugin, it can be specified as 0.

- deinit**

A once-only deinitialization function. This is executed when the plugin is unloaded, which happens for `UNINSTALL PLUGIN` or, for plugins listed in the `plugin` table, at server shutdown. The function takes no arguments. It returns zero for success and non-zero for failure. If a `deinit` function is unneeded for a plugin, it can be specified as 0.

- version**

The plugin version number. When the plugin is installed, this value can be retrieved from the `INFORMATION_SCHEMA.PLUGINS` table. The value includes major and minor numbers. If you write the value as a hex constant, the format is `0xMMNN`, where `MM` and `NN` are the major and minor numbers, respectively. For example, `0x0302` represents version 3.2.

- `status_vars`

A pointer to a structure for status variables associated with the plugin, or 0 if there are no such variables. When the plugin is installed, these variables are displayed in the output of the `SHOW STATUS` statement.

- `__reserved1, __reserved2`

These are placeholders for the future. Currently, they should be set to `NULL`.

The `init` and `deinit` functions in the general plugin declaration are invoked only when loading and unloading the plugin. They have nothing to do with use of the plugin such as happens when an SQL statement causes the plugin to be invoked.

The `status_vars` member, if not 0, points to an array of `st_mysql_show_var` structures, each of which describes one status variable, followed by a structure with all members set to 0. The `st_mysql_show_var` structure has this definition:

```
struct st_mysql_show_var {
    const char *name;
    char *value;
    enum enum_mysql_show_type type;
};
```

When the plugin is installed, the plugin name and the `name` value are joined with an underscore to form the name displayed by `SHOW STATUS`.

The following table shows the allowable status variable `type` values and what the corresponding variable should be:

Type	Meaning
<code>SHOW_BOOL</code>	Pointer to a boolean variable
<code>SHOW_INT</code>	Pointer to an integer variable
<code>SHOW_LONG</code>	Pointer to a long integer variable
<code>SHOW_LOONGLONG</code>	Pointer to a longlong integer variable
<code>SHOW_CHAR</code>	A string
<code>SHOW_CHAR_PTR</code>	Pointer to a string
<code>SHOW_ARRAY</code>	Pointer to another <code>st_mysql_show_var</code> array
<code>SHOW_FUNC</code>	Pointer to a function

For the `SHOW_FUNC` type, the function is called and fills in its `out` parameter, which then provides information about the variable to be displayed. The function has this calling sequence:

```
#define SHOW_VAR_FUNC_BUFF_SIZE 1024
typedef int (*mysql_show_var_func) (void *thd,
                                    struct st_mysql_show_var *out,
                                    char *buf);
```

Plugins should consider the `thd` parameter to be read-only.

28.2.5.2. Type-Specific Plugin Structures and Functions

In the `st_mysql_plugin` structure that defines a plugin's general declaration, the `info` member points to a type-specific plugin descriptor. For a full-text parser plugin, the descriptor corresponds to the `st_mysql_ftparser` structure in the `plugin.h` file:

```
struct st_mysql_ftparser
{
    int interface_version;
    int (*parse)(MYSQL_FTPARSER_PARAM *param);
    int (*init)(MYSQL_FTPARSER_PARAM *param);
    int (*deinit)(MYSQL_FTPARSER_PARAM *param);
};
```

```
};
```

As shown by the structure definition, the descriptor has a version number (`MYSQL_FTPARSER_INTERFACE_VERSION` for full-text parser plugins) and contains pointers to three functions. The `init` and `deinit` members should point to a function or be set to 0 if the function is not needed. The `parse` member must point to the function that performs the parsing.

A full-text parser plugin is used in two different contexts, indexing and searching. In both contexts, the server calls the initialization and deinitialization functions at the beginning and end of processing each SQL statement that causes the plugin to be invoked. However, during statement processing, the server calls the main parsing function in context-specific fashion:

- For indexing, the server calls the parser for each column value to be indexed.
- For searching, the server calls the parser to parse the search string. The parser might also be called for rows processed by the statement. In natural language mode, there is no need for the server to call the parser. For boolean mode phrase searches or natural language searches with query expansion, the parser is used to parse column values for information that is not in the index. Also, if a boolean mode search is done for a column that has no `FULLTEXT` index, the built-in parser will be called. (Plugins are associated with specific indexes. If there is no index, no plugin is used.)

Note that the plugin declaration in the plugin library descriptor has initialization and deinitialization functions, and so does the plugin descriptor to which it points. These pairs of functions have different purposes and are invoked for different reasons:

- For the plugin declaration in the plugin library descriptor, the initialization and deinitialization functions are invoked when the plugin is loaded and unloaded.
- For the plugin descriptor, the initialization and deinitialization functions are invoked per SQL statement for which the plugin is used.

Each interface function named in the plugin descriptor should return zero for success or non-zero for failure, and each of them receives an argument that points to a `MYSQL_FTPARSER_PARAM` structure containing the parsing context. The structure has this definition:

```
typedef struct st_mysql_ftparser_param
{
    int (*mysql_parse)(struct st_mysql_ftparser_param *,
                      char *doc, int doc_len);
    int (*mysql_add_word)(struct st_mysql_ftparser_param *,
                        char *word, int word_len,
                        MYSQL_FTPARSER_BOOLEAN_INFO *boolean_info);
    void *ftparser_state;
    void *mysql_ftparam;
    struct charset_info_st *cs;
    char *doc;
    int length;
    int flags;
    enum enum_ftparser_mode mode;
} MYSQL_FTPARSER_PARAM;
```

Note

The definition shown is current as of MySQL 5.1.12. It is incompatible with versions of MySQL 5.1 older than 5.1.12.

The structure members are used as follows:

- `mysql_parse`

A pointer to a callback function that invokes the server's built-in parser. Use this callback when the plugin acts as a front end to the built-in parser. That is, when the plugin parsing function is called, it should process the input to extract the text and pass the text to the `mysql_parse` callback.

The first parameter for this callback function should be the `param` value itself:

```
param->mysql_parse(param, ...);
```

A front end plugin can extract text and pass it all at once to the built-in parser, or it can extract and pass text to the built-in parser a

piece at a time. However, in this case, the built-in parser treats the pieces of text as though there are implicit word breaks between them.

- `mysql_add_word`

A pointer to a callback function that adds a word to a full-text index or to the list of search terms. Use this callback when the parser plugin replaces the built-in parser. That is, when the plugin parsing function is called, it should parse the input into words and invoke the `mysql_add_word` callback for each word.

The first parameter for this callback function should be the `param` value itself:

```
param->mysql_add_word(param, ...);
```

- `ftparser_state`

This is a generic pointer. The plugin can set it to point to information to be used internally for its own purposes.

- `mysql_ftparam`

This is set by the server. It is passed as the first argument to the `mysql_parse` or `mysql_add_word` callback.

- `cs`

A pointer to information about the character set of the text, or 0 if no information is available.

- `doc`

A pointer to the text to be parsed.

- `length`

The length of the text to be parsed, in bytes.

- `flags`

Parser flags. This is zero if there are no special flags. Currently, the only non-zero flag is `MYSQL_FTFIELDS_NEED_COPY`, which means that `mysql_add_word()` must save a copy of the word (that is, it cannot use a pointer to the word because the word is in a buffer that will be overwritten.) This member was added in MySQL 5.1.12.

This flag might be set or reset by MySQL before calling the parser plugin, by the parser plugin itself, or by the `mysql_parse()` function.

- `mode`

The parsing mode. This value will be one of the following constants:

- `MYSQL_FTPARSER_SIMPLE_MODE`

Parse in fast and simple mode, which is used for indexing and for natural language queries. The parser should pass to the server only those words that should be indexed. If the parser uses length limits or a stopword list to determine which words to ignore, it should not pass such words to the server.

- `MYSQL_FTPARSER_WITH_STOPWORDS`

Parse in stopword mode. This is used in boolean searches for phrase matching. The parser should pass all words to the server, even stopwords or words that are outside any normal length limits.

- `MYSQL_FTPARSER_FULL_BOOLEAN_INFO`

Parse in boolean mode. This is used for parsing boolean query strings. The parser should recognize not only words but also boolean-mode operators and pass them to the server as tokens via the `mysql_add_word` callback. To tell the server what kind of token is being passed, the plugin needs to fill in a `MYSQL_FTPARSER_BOOLEAN_INFO` structure and pass a pointer to it.

If the parser is called in boolean mode, the `param->mode` value will be `MYSQL_FTPARSER_FULL_BOOLEAN_INFO`. The

`MYSQL_FTPARSER_BOOLEAN_INFO` structure that the parser uses for passing token information to the server looks like this:

```
typedef struct st_mysql_ftparser_boolean_info
{
    enum enum_ft_token_type type;
    int yesno;
    int weight_adjust;
    bool wasign;
    bool trunc;
    /* These are parser state and must be removed. */
    byte prev;
    byte *quot;
} MYSQL_FTPARSER_BOOLEAN_INFO;
```

The parser should fill in the structure members as follows:

- `type`

The token type. This should be one of values shown in the following table:

Type	Meaning
<code>FT_TOKEN_EOF</code>	End of data
<code>FT_TOKEN_WORD</code>	A regular word
<code>FT_TOKEN_LEFT_PAREN</code>	The beginning of a group or subexpression
<code>FT_TOKEN_RIGHT_PAREN</code>	The end of a group or subexpression
<code>FT_TOKEN_STOPWORD</code>	A stopword

- `yesno`

Whether the word must be present for a match to occur. 0 means that the word is optional but increases the match relevance if it is present. Values larger than 0 mean that the word must be present. Values smaller than 0 mean that the word must not be present.

- `weight_adjust`

A weighting factor that determines how much a match for the word counts. It can be used to increase or decrease the word's importance in relevance calculations. A value of zero indicates no weight adjustment. Values greater than or less than zero mean higher or lower weight, respectively. The examples at [Section 11.8.1, “Boolean Full-Text Searches”](#), that use the `<` and `>` operators illustrate how weighting works.

- `wasign`

The sign of the weighting factor. A negative value acts like the `~` boolean-search operator, which causes the word's contribution to the relevance to be negative.

- `trunc`

Whether matching should be done as if the boolean-mode `*` truncation operator had been given.

Plugins should not use the `prev` and `quot` members of the `MYSQL_FTPARSER_BOOLEAN_INFO` structure.

28.2.5.3. Creating a Plugin Library

This section provides a step-by-step procedure for creating a plugin library. It shows how to develop a library that contains a full-text parsing plugin named `simple_parser`. This plugin performs parsing based on simpler rules than those used by the MySQL built-in full-text parser: Words are non-empty runs of whitespace characters.

Each plugin library has the following contents:

- A plugin library descriptor that indicates the version number of the general plugin API that the library uses and that contains a general declaration for each plugin in the library.
- Each plugin general declaration contains information that is common to all types of plugin: A value that indicates the plugin type;

the plugin name, author, description, and license type; and pointers to the initialization and deinitialization functions that the server invokes when it loads and unloads the plugin.

- The plugin general declaration also contains a pointer to a type-specific plugin descriptor. The structure of these descriptors can vary from one plugin type to another, because each type of plugin can have its own API. A plugin descriptor contains a type-specific API version number and pointers to the functions that are needed to implement that plugin type. For example, a full-text parser plugin has initialization and deinitialization functions, and a main parsing function. The server invokes these functions when it uses the plugin to parse text.
- The plugin library contains the interface functions that are referenced by the library descriptor and by the plugin descriptors.

The easiest way to follow the instructions in this section is to use the source code in the `plugin/fulltext` directory of a MySQL source distribution. The instructions assume that you make a copy of that directory and use it to build the plugin library. To make a copy of the directory, use the following commands, which assume that the MySQL source tree is in a directory named `mysql-5.1` under your current directory:

```
shell> mkdir fulltext_plugin
shell> cp mysql-5.1/plugin/fulltext/* fulltext_plugin
```

If you are copying files from a BitKeeper source tree, `cp` will display an error message about the `SCCS` directory, which you can ignore.

After copying the source files, use the following procedure to create a plugin library:

1. Change location into the `fulltext_plugin` directory:

```
shell> cd fulltext_plugin
```

2. The plugin source file should include the header files that the plugin library needs. The `plugin.h` file is required, and the library might require other files as well. For example:

```
#include <stdlib.h>
#include <ctype.h>
#include <mysql/plugin.h>
```

3. Set up the plugin library file descriptor.

Every plugin library must include a library descriptor that must define two symbols:

- `_mysql_plugin_interface_version_` specifies the version number of the general plugin framework. This is given by the `MYSQL_PLUGIN_INTERFACE_VERSION` symbol, which is defined in the `plugin.h` file.
- `_mysql_plugin_declarations_` defines an array of plugin declarations, terminated by a declaration with all members set to 0. Each declaration is an instance of the `st_mysql_plugin` structure (also defined in `plugin.h`). There must be one of these for each plugin in the library.

If the server does not find these two symbols in a library, it does not accept it as a legal plugin library and rejects it with an error. This prevents use of a library for plugin purposes unless it was built specifically as a plugin library.

The standard (and most convenient) way to define the two required symbols is by using the `mysql_declare_plugin` and `mysql_declare_plugin_end` macros from the `plugin.h` file:

```
mysql_declare_plugin
... one or more plugin declarations here ...
mysql_declare_plugin_end;
```

For example, the library descriptor for a library that contains a single plugin named `simple_parser` looks like this:

```
mysql_declare_plugin
{
    MYSQL_FTPARSER_PLUGIN,      /* type */
    &simple_parser_descriptor,    /* descriptor */
    "simple_parser",             /* name */
    "MySQL AB",                 /* author */
    "Simple Full-Text Parser",   /* description */
    PLUGIN_LICENSE_GPL,         /* plugin license */
}
```

```

    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status /* status variables */
}
mysql_declare_plugin_end;

```

For a full-text parser plugin, the type must be `MYSQL_FTPARSER_PLUGIN`. This is the value that identifies the plugin as being legal for use in a `WITH PARSER` clause when creating a `FULLTEXT` index. (No other plugin type is legal for this clause.)

The `mysql_declare_plugin` and `mysql_declare_plugin_end` macros are defined in `plugin.h` like this:

```

#ifndef MYSQL_DYNAMIC_PLUGIN
#define __MYSQL_DECLARE_PLUGIN(NAME, VERSION, PSIZE, DECLS) \
int VERSION= MYSQL_PLUGIN_INTERFACE_VERSION; \
int PSIZE= sizeof(struct st_mysql_plugin); \
struct st_mysql_plugin DECLS[]= { \
#else
#define __MYSQL_DECLARE_PLUGIN(NAME, VERSION, PSIZE, DECLS) \
int _mysql_plugin_interface_version= MYSQL_PLUGIN_INTERFACE_VERSION; \
int _mysql_sizeof_struct_st_plugin= sizeof(struct st_mysql_plugin); \
struct st_mysql_plugin _mysql_plugin_declarations[]= { \
#endif

#define mysql_declare_plugin(NAME) \
__MYSQL_DECLARE_PLUGIN(NAME, \
    builtin_ ## NAME ## _plugin_interface_version, \
    builtin_ ## NAME ## _sizeof_struct_st_plugin, \
    builtin_ ## NAME ## _plugin)

#define mysql_declare_plugin_end , {0,0,0,0,0,0,0,0,0}}

```

One point to note about those definitions is that the `_mysql_plugin_interface_version` symbol is defined only if the `MYSQL_DYNAMIC_PLUGIN` symbol is defined. This means that you'll need to provide `-DMYSQL_DYNAMIC_PLUGIN` as part of the compilation command when you build the plugin.

When the macros are used as just shown, they expand to the following code, which defines both of the required symbols (`_mysql_plugin_interface_version` and `_mysql_plugin_declarations`):

```

int _mysql_plugin_interface_version= MYSQL_PLUGIN_INTERFACE_VERSION;
struct st_mysql_plugin _mysql_plugin_declarations[]= {
{
    MYSQL_FTPARSER_PLUGIN, /* type */
    &simple_parser_descriptor, /* descriptor */
    "simple_parser", /* name */
    "MySQL AB", /* author */
    "Simple Full-Text Parser", /* description */
    PLUGIN_LICENSE_GPL, /* plugin license */
    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status /* status variables */
}
, {0,0,0,0,0,0,0,0,0}}
};

```

The preceding example declares a single plugin in the library descriptor, but it is possible to declare multiple plugins. List the declarations one after the other between `mysql_declare_plugin` and `mysql_declare_plugin_end`, separated by commas.

MySQL plugins can be written in C or C++ (or another language that can use C calling conventions). One feature of C++ is that you can use non-constant variables to initialize global structures. However, if you write a C++ plugin, you should not use this feature. Members of structures such as the `st_mysql_plugin` structure should be initialized with constant variables. See the discussion at the end of this section that describes some legal and illegal initializers for plugins.

4. Set up the plugin descriptor.

Each plugin declaration in the library descriptor points to a type-specific descriptor for the corresponding plugin. In the `simple_parser` declaration, that descriptor is indicated by `&simple_parser_descriptor`. The descriptor specifies the version number for the full-text plugin interface (as given by `MYSQL_FTPARSER_INTERFACE_VERSION`), and the plugin's parsing, initialization, and deinitialization functions:

```

static struct st_mysql_ftparser simple_parser_descriptor=
{

```

```

MYSQL_FTPARSER_INTERFACE_VERSION, /* interface version */
simple_parser_parse,               /* parsing function */
simple_parser_init,                /* parser init function */
simple_parser_deinit               /* parser deinit function */
};

```

5. Set up the plugin interface functions.

The general plugin declaration in the library descriptor names the initialization and deinitialization functions that the server should invoke when it loads and unloads the plugin. For `simple_parser`, these functions do nothing but return zero to indicate that they succeeded:

```

static int simple_parser_plugin_init(void)
{
    return(0);
}

static int simple_parser_plugin_deinit(void)
{
    return(0);
}

```

Because those functions do not actually do anything, you could omit them and specify 0 for each of them in the plugin declaration.

The type-specific plugin descriptor for `simple_parser` names the initialization, deinitialization, and parsing functions that the server invokes when the plugin is used. For `simple_parser`, the initialization and deinitialization functions do nothing:

```

static int simple_parser_init(MYSQL_FTPARSER_PARAM *param)
{
    return(0);
}

static int simple_parser_deinit(MYSQL_FTPARSER_PARAM *param)
{
    return(0);
}

```

Here too, because those functions do nothing, you could omit them and specify 0 for each of them in the plugin descriptor.

The main parsing function, `simple_parser_parse()`, acts as a replacement for the built-in full-text parser, so it needs to split text into words and pass each word to the server. The parsing function's first argument is a pointer to a structure that contains the parsing context. This structure has a `doc` member that points to the text to be parsed, and a `length` member that indicates how long the text is. The simple parsing done by the plugin considers non-empty runs of whitespace characters to be words, so it identifies words like this:

```

static int simple_parser_parse(MYSQL_FTPARSER_PARAM *param)
{
    char *end, *start, *docend= param->doc + param->length;
    for (end= start= param->doc;; end++)
    {
        if (end == docend)
        {
            if (end > start)
                add_word(param, start, end - start);
            break;
        }
        else if (isspace(*end))
        {
            if (end > start)
                add_word(param, start, end - start);
            start= end + 1;
        }
    }
    return(0);
}

```

As the parser finds each word, it invokes a function `add_word()` to pass the word to the server. `add_word()` is a helper function only; it is not part of the plugin interface. The parser passes the parsing context pointer to `add_word()`, as well as a pointer to the word and a length value:

```

static void add_word(MYSQL_FTPARSER_PARAM *param, char *word, size_t len)
{
    MYSQL_FTPARSER_BOOLEAN_INFO bool_info=
        { FT_TOKEN_WORD, 0, 0, 0, 0, '-', 0 };
}

```

```

    param->mysql_add_word(param, word, len, &bool_info);
}
    
```

For boolean-mode parsing, `add_word()` fills in the members of the `bool_info` structure as described in [Section 28.2.5.2](#), “Type-Specific Plugin Structures and Functions”.

- Set up the status variables, if there are any. For the `simple_parser` plugin, the following status variable array sets up one status variable with a value that is static text, and another with a value that is stored in a long integer variable:

```

long number_of_calls= 0;

struct st_mysql_show_var simple_status[]=
{
    {"static", (char *)"just a static text", SHOW_CHAR},
    {"called", (char *)&number_of_calls, SHOW_LONG},
    {0,0,0}
};
    
```

When the plugin is installed, the plugin name and the `name` value are joined with an underscore to form the name displayed by `SHOW STATUS`. For the array just shown, the resulting status variable names are `simple_parser_static` and `simple_parser_called`. This convention means that you can easily display the variables for a plugin using its name:

```

mysql> SHOW STATUS LIKE 'simple_parser%';
+-----+-----+
| Variable_name | Value               |
+-----+-----+
| simple_parser_static | just a static text |
| simple_parser_called | 0                   |
+-----+-----+
    
```

- Compile the plugin library as a shared library and install it in the plugin directory.

Note

As mentioned earlier, be sure to specify `-DMYSQL_DYNAMIC_PLUGIN` as part of the compilation command when you build the plugin.

The procedure for compiling shared objects varies from system to system. If you build your library using the GNU autotools, `libtool` should be able to generate the correct compilation commands for your system. If the library is named `mypluglib`, you should end up with a shared object file that has a name something like `libmypluglib.so`. (The filename might have a different extension on your system.)

To use the autotools, you'll need to make a few changes to the configuration files at this point to enable the plugin to be compiled and installed. Assume that your MySQL distribution is installed at a base directory of `/usr/local/mysql` and that its header files are located in the `include` directory under the base directory.

Edit `Makefile.am`, which should look something like this:

```

#Makefile.am example for a plugin

pkglibdir=$(libdir)/mysql
INCLUDES= -I$(top_builddir)/include -I$(top_srcdir)/include
#noinst_LTLIBRARIES= mypluglib_la
pkglib_LTLIBRARIES= mypluglib_la
mypluglib_la_SOURCES= plugin_example.c
mypluglib_la_LDFLAGS= -module -rpath $(pkglibdir)
mypluglib_la_CFLAGS= -DMYSQL_DYNAMIC_PLUGIN
    
```

The `mypluglib_la_CFLAGS` line takes care of passing the `-DMYSQL_DYNAMIC_PLUGIN` flag to the compilation command.

Adjust the `INCLUDES` line to specify the pathname to the installed MySQL header files. Edit it to look like this:

```

INCLUDES= -I/usr/local/mysql/include
    
```

Make sure that the `noinst_LTLIBRARIES` line is commented out or remove it. Make sure that the `pkglib_LTLIBRARIES` line is not commented out; it enables the `make install` command.

Set up the files needed for the `configure` command, invoke it, and run `make`:

```
shell> autoreconf --force --install --symlink
shell> ./configure --prefix=/usr/local/mysql
shell> make
```

The `--prefix` option to `configure` indicates the MySQL base directory under which the plugin should be installed. You can see what value to use for this option with `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'basedir';
+-----+-----+
| Variable_name | Value                |
+-----+-----+
| base          | /usr/local/mysql    |
+-----+-----+
```

The location of the plugin directory where you should install the library is given by the `plugin_dir` system variable. For example:

```
mysql> SHOW VARIABLES LIKE 'plugin_dir';
+-----+-----+
| Variable_name | Value                |
+-----+-----+
| plugin_dir    | /usr/local/mysql/lib/mysql |
+-----+-----+
```

To install the plugin library, use `make`:

```
shell> make install
```

Verify that `make install` installed the plugin library in the proper directory. After installing it, make sure that the library permissions allow it to be executed by the server.

8. Register the plugin with the server.

The `INSTALL PLUGIN` statement causes the server to list the plugin in the `plugin` table and to load the plugin code from the library file. Use that statement to register `simple_parser` with the server, and then verify that the plugin is listed in the `plugin` table:

```
mysql> INSTALL PLUGIN simple_parser SONAME 'libmypluglib.so';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM mysql.plugin;
+-----+-----+
| name      | dl                |
+-----+-----+
| simple_parser | libmypluglib.so |
+-----+-----+
1 row in set (0.00 sec)
```

9. Try the plugin.

Create a table that contains a string column and associate the parser plugin with a `FULLTEXT` index on the column:

```
mysql> CREATE TABLE t (c VARCHAR(255),
-> FULLTEXT (c) WITH PARSER simple_parser);
Query OK, 0 rows affected (0.01 sec)
```

Insert some text into the table and try some searches. These should verify that the parser plugin treats all non-whitespace characters as word characters:

```
mysql> INSERT INTO t VALUES
-> ('latin1_general_cs is a case-sensitive collation'),
-> ('I\'d like a case of oranges'),
-> ('this is sensitive information'),
-> ('another row'),
-> ('yet another row');
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT c FROM t;
+-----+
| c                |
+-----+
```

```
| latin1_general_cs is a case-sensitive collation |
| I'd like a case of oranges                     |
| this is sensitive information                  |
| another row                                   |
| yet another row                               |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT MATCH(c) AGAINST('case') FROM t;
+-----+
| MATCH(c) AGAINST('case') |
+-----+
|                          |
|          0               |
| 1.2968142032623         |
|          0               |
|          0               |
|          0               |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT MATCH(c) AGAINST('sensitive') FROM t;
+-----+
| MATCH(c) AGAINST('sensitive') |
+-----+
|                                |
|                                |
|          0                   |
| 1.3253291845322             |
|          0                   |
|          0                   |
|          0                   |
+-----+
5 rows in set (0.01 sec)

mysql> SELECT MATCH(c) AGAINST('case-sensitive') FROM t;
+-----+
| MATCH(c) AGAINST('case-sensitive') |
+-----+
|                                     |
|                                     |
|          0                         |
| 1.3109166622162                 |
|          0                         |
|          0                         |
|          0                         |
+-----+
5 rows in set (0.01 sec)

mysql> SELECT MATCH(c) AGAINST('I\'d') FROM t;
+-----+
| MATCH(c) AGAINST('I\'d') |
+-----+
|                          |
|          0               |
| 1.2968142032623         |
|          0               |
|          0               |
|          0               |
+-----+
5 rows in set (0.01 sec)
```

Note how neither “case” nor “insensitive” match “case-insensitive” the way that they would for the built-in parser.

MySQL plugins can be written in C or C++ (or another language that can use C calling conventions). One feature of C++ is that you can use non-constant variables to initialize global structures. However, if you write a C++ plugin, you should not use this feature. Members of structures such as the `st_mysql_plugin` structure should be initialized with constant variables. The `simple_parser` descriptor shown earlier is allowable in a C++ plugin because it satisfies that requirement:

```
mysql_declare_plugin
{
    MYSQL_FTPARSER_PLUGIN, /* type */
    &simple_parser_descriptor, /* descriptor */
    "simple_parser", /* name */
    "MySQL AB", /* author */
    "Simple Full-Text Parser", /* description */
    PLUGIN_LICENSE_GPL, /* plugin license */
    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status /* status variables */
}
mysql_declare_plugin_end;
```

Here is another valid way to write the descriptor. It uses constant variables to indicate the plugin name, author, and description:

```
const char *simple_parser_name = "simple_parser";
const char *simple_parser_author = "MySQL AB";
```



```
const char *simple_parser_description = "Simple Full-Text Parser";

mysql_declare_plugin
{
    MYSQL_FTPARSER_PLUGIN, /* type */
    &simple_parser_descriptor, /* descriptor */
    simple_parser_name, /* name */
    simple_parser_author, /* author */
    simple_parser_description, /* description */
    PLUGIN_LICENSE_GPL, /* plugin license */
    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status /* status variables */
}
mysql_declare_plugin_end;
```

However, the following descriptor is invalid. It uses structure members to indicate the plugin name, author, and description, but structures are not considered constant initializers in C++:

```
typedef struct
{
    const char *name;
    const char *author;
    const char *description;
} plugin_info;

plugin_info parser_info = {
    "simple_parser",
    "MySQL AB",
    "Simple Full-Text Parser"
};

mysql_declare_plugin
{
    MYSQL_FTPARSER_PLUGIN, /* type */
    &simple_parser_descriptor, /* descriptor */
    parser_info.name, /* name */
    parser_info.author, /* author */
    parser_info.description, /* description */
    PLUGIN_LICENSE_GPL, /* plugin license */
    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status /* status variables */
}
mysql_declare_plugin_end;
```

28.3. Adding New Functions to MySQL

There are three ways to add new functions to MySQL:

- You can add functions through the user-defined function (UDF) interface. User-defined functions are compiled as object files and then added to and removed from the server dynamically using the [CREATE FUNCTION](#) and [DROP FUNCTION](#) statements. See [Section 28.3.2, “CREATE FUNCTION Syntax”](#).
- You can add functions as native (built-in) MySQL functions. Native functions are compiled into the `mysqld` server and become available on a permanent basis.
- Another way to add functions is by creating stored functions. These are written using SQL statements rather than by compiling object code. The syntax for writing stored functions is not covered here. See [Chapter 19, Stored Procedures and Functions](#).

Each method of creating compiled functions has advantages and disadvantages:

- If you write user-defined functions, you must install object files in addition to the server itself. If you compile your function into the server, you don't need to do that.
- Native functions require you to modify a source distribution. UDFs do not. You can add UDFs to a binary MySQL distribution. No access to MySQL source is necessary.
- If you upgrade your MySQL distribution, you can continue to use your previously installed UDFs, unless you upgrade to a newer version for which the UDF interface changes. For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they can be invoked in SQL statements just like native functions such as `ABS()` or `SOUNDEX()`.

See [Section 8.2.4, “Function Name Parsing and Resolution”](#), for the rules describing how the server interprets references to different kinds of functions.

The following sections describe features of the UDF interface, provide instructions for writing UDFs, discuss security precautions that MySQL takes to prevent UDF misuse, and describe how to add native MySQL functions.

For example source code that illustrates how to write UDFs, take a look at the `sql/udf_example.c` file that is provided in MySQL source distributions.

28.3.1. Features of the User-Defined Function Interface

The MySQL interface for user-defined functions provides the following features and capabilities:

- Functions can return string, integer, or real values and can accept arguments of those same types.
- You can define simple functions that operate on a single row at a time, or aggregate functions that operate on groups of rows.
- Information is provided to functions that enables them to check the number, types, and names of the arguments passed to them.
- You can tell MySQL to coerce arguments to a given type before passing them to a function.
- You can indicate that a function returns `NULL` or that an error occurred.

28.3.2. CREATE FUNCTION Syntax

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|INTEGER|REAL|DECIMAL}
SONAME shared_library_name
```

A user-defined function (UDF) is a way to extend MySQL with a new function that works like a native (built-in) MySQL function such as `ABS()` or `CONCAT()`.

function_name is the name that should be used in SQL statements to invoke the function. The `RETURNS` clause indicates the type of the function's return value. `DECIMAL` is a legal value after `RETURNS`, but currently `DECIMAL` functions return string values and should be written like `STRING` functions.

shared_library_name is the basename of the shared object file that contains the code that implements the function. The file must be located in the plugin directory. This directory is given by the value of the `plugin_dir` system variable. (**Note:** This a change in MySQL 5.1. For earlier versions of MySQL, the shared object can be located in any directory that is searched by your system's dynamic linker.)

To create a function, you must have the `INSERT` privilege for the `mysql` database. This is necessary because `CREATE FUNCTION` adds a row to the `mysql.func` system table that records the function's name, type, and shared library name. If you do not have this table, you should run the `mysql_upgrade` command to create it. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable.

For instructions on writing user-defined functions, see [Section 28.3.4, “Adding a New User-Defined Function”](#). For the UDF mechanism to work, functions must be written in C or C++ (or another language that can use C calling conventions), your operating system must support dynamic loading and you must have compiled `mysqld` dynamically (not statically).

An `AGGREGATE` function works exactly like a native MySQL aggregate (summary) function such as `SUM` or `COUNT()`. For `AGGREGATE` to work, your `mysql.func` table must contain a `type` column. If your `mysql.func` table does not have this column, you should run the `mysql_upgrade` program to create it (see [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#)).

28.3.3. DROP FUNCTION Syntax

```
DROP FUNCTION function_name
```

This statement drops the user-defined function (UDF) named *function_name*.

To drop a function, you must have the **DELETE** privilege for the **mysql** database. This is because **DROP FUNCTION** removes a row from the **mysql.func** system table that records the function's name, type, and shared library name.

28.3.4. Adding a New User-Defined Function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. The MySQL source distribution includes a file **sql/udf_example.c** that defines 5 new functions. Consult this file to see how UDF calling conventions work. UDF-related symbols and data structures are defined in the **include/mysql_com.h** header file. (You need not include this header file directly because it is included by **mysql.h**.)

A UDF contains code that becomes part of the running server, so when you write a UDF, you are bound by any and all constraints that otherwise apply to writing server code. For example, you may have problems if you attempt to use functions from the **libstdc++** library. These constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to UDFs that were originally written for older servers. For information about these constraints, see [Section 2.9.2, “Typical configure Options”](#), and [Section 2.9.4, “Dealing with Problems Compiling MySQL”](#).

To be able to use UDFs, you need to link **mysqld** dynamically. Don't configure MySQL using **-with-mysqld-ldflags=-all-static**. If you want to use a UDF that needs to access symbols from **mysqld** (for example, the **metaphone** function in **sql/udf_example.c** that uses **default_charset_info**), you must link the program with **-rdynamic** (see [man dlopen](#)). If you plan to use UDFs, the rule of thumb is to configure MySQL with **-with-mysqld-ldflags=-rdynamic** unless you have a very good reason not to.

For each function that you want to use in SQL statements, you should define corresponding C (or C++) functions. In the following discussion, the name “xxx” is used for an example function name. To distinguish between SQL and C/C++ usage, **XXX()** (uppercase) indicates an SQL function call, and **xxx()** (lowercase) indicates a C/C++ function call.

The C/C++ functions that you write to implement the interface for **xxx()** are:

- **xxx()** (required)

The main function. This is where the function result is computed. The correspondence between the SQL function data type and the return type of your C/C++ function is shown here:

SQL Type	C/C++ Type
STRING	char *
INTEGER	long long
REAL	double

It is also possible to declare a **DECIMAL** function, but currently the value is returned as a string, so you should write the UDF as though it were a **STRING** function. **ROW** functions are not implemented.

- **xxx_init()** (optional)

The initialization function for **xxx()**. It can be used for the following purposes:

- To check the number of arguments to **XXX()**.
- To check that the arguments are of a required type or, alternatively, to tell MySQL to coerce arguments to the types you want when the main function is called.
- To allocate any memory required by the main function.
- To specify the maximum length of the result.
- To specify (for **REAL** functions) the maximum number of decimal places in the result.
- To specify whether the result can be **NULL**.

- `xxx_deinit()` (optional)

The deinitialization function for `xxx()`. It should deallocate any memory allocated by the initialization function.

When an SQL statement invokes `XXX()`, MySQL calls the initialization function `xxx_init()` to let it perform any required setup, such as argument checking or memory allocation. If `xxx_init()` returns an error, MySQL aborts the SQL statement with an error message and does not call the main or deinitialization functions. Otherwise, MySQL calls the main function `xxx()` once for each row. After all rows have been processed, MySQL calls the deinitialization function `xxx_deinit()` so that it can perform any required cleanup.

For aggregate functions that work like `SUM()`, you must also provide the following functions:

- `xxx_clear()` (required in 5.1)

Reset the current aggregate value but do not insert the argument as the initial aggregate value for a new group.

- `xxx_add()` (required)

Add the argument to the current aggregate value.

MySQL handles aggregate UDFs as follows:

1. Call `xxx_init()` to let the aggregate function allocate any memory it needs for storing results.
2. Sort the table according to the `GROUP BY` expression.
3. Call `xxx_clear()` for the first row in each new group.
4. Call `xxx_add()` for each new row that belongs in the same group.
5. Call `xxx()` to get the result for the aggregate when the group changes or after the last row has been processed.
6. Repeat 3-5 until all rows has been processed
7. Call `xxx_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe. This includes not just the main function, but the initialization and deinitialization functions as well, and also the additional functions required by aggregate functions. A consequence of this requirement is that you are not allowed to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

28.3.4.1. UDF Calling Sequences for Simple Functions

This section describes the different functions that you need to define when you create a simple UDF. [Section 28.3.4, “Adding a New User-Defined Function”](#), describes the order in which MySQL calls these functions.

The main `xxx()` function should be declared as shown in this section. Note that the return type and parameters differ, depending on whether you declare the SQL function `XXX()` to return `STRING`, `INTEGER`, or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
           char *is_null, char *error);
```

DECIMAL functions return string values and should be declared the same way as **STRING** functions. **ROW** functions are not implemented.

The initialization and deinitialization functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
void xxx_deinit(UDF_INIT *initid);
```

The **initid** parameter is passed to all three functions. It points to a **UDF_INIT** structure that is used to communicate information between functions. The **UDF_INIT** structure members follow. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

- **my_bool maybe_null**

xxx_init() should set **maybe_null** to 1 if **xxx()** can return **NULL**. The default value is 1 if any of the arguments are declared **maybe_null**.

- **unsigned int decimals**

The number of decimal digits to the right of the decimal point. The default value is the maximum number of decimal digits in the arguments passed to the main function. (For example, if the function is passed **1.34**, **1.345**, and **1.3**, the default would be 3, because **1.345** has 3 decimal digits.

- **unsigned int max_length**

The maximum length of the result. The default **max_length** value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimal digits indicated by **initid->decimals**. (For numeric functions, the length includes any sign or decimal point characters.)

If you want to return a blob value, you can set **max_length** to 65KB or 16MB. This memory is not allocated, but the value is used to decide which data type to use if there is a need to temporarily store the data.

- **char *ptr**

A pointer that the function can use for its own purposes. For example, functions can use **initid->ptr** to communicate allocated memory among themselves. **xxx_init()** should allocate the memory and assign it to this pointer:

```
initid->ptr = allocated_memory;
```

In **xxx()** and **xxx_deinit()**, refer to **initid->ptr** to use or deallocate the memory.

- **my_bool const_item**

xxx_init() should set **const_item** to 1 if **xxx()** always returns the same value and to 0 otherwise.

28.3.4.2. UDF Calling Sequences for Aggregate Functions

This section describes the different functions that you need to define when you create an aggregate UDF. [Section 28.3.4, “Adding a New User-Defined Function”](#), describes the order in which MySQL calls these functions.

- **xxx_reset()**

This function is called when MySQL finds the first row in a new group. It should reset any internal summary variables and then use the given **UDF_ARGS** argument as the first value in your internal summary value for the group. Declare **xxx_reset()** as follows:

```
char *xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
                char *is_null, char *error);
```

`xxx_reset()` is not needed or used in MySQL 5.1, in which the UDF interface uses `xxx_clear()` instead. However, you can define both `xxx_reset()` and `xxx_clear()` if you want to have your UDF work with older versions of the server. (If you do include both functions, the `xxx_reset()` function in many cases can be implemented internally by calling `xxx_clear()` to reset all variables, and then calling `xxx_add()` to add the `UDF_ARGS` argument as the first value in the group.)

- `xxx_clear()`

This function is called when MySQL needs to reset the summary results. It is called at the beginning for each new group but can also be called to reset the values for a query where there were no matching rows. Declare `xxx_clear()` as follows:

```
char *xxx_clear(UDF_INIT *initid, char *is_null, char *error);
```

`is_null` is set to point to `CHAR(0)` before calling `xxx_clear()`.

If something went wrong, you can store a value in the variable to which the `error` argument points. `error` points to a single-byte variable, not to a string buffer.

`xxx_clear()` is required by MySQL 5.1.

- `xxx_add()`

This function is called for all rows that belong to the same group, except for the first row. You should use it to add the value in the `UDF_ARGS` argument to your internal summary variable.

```
char *xxx_add(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

The `xxx()` function for an aggregate UDF should be declared the same way as for a non-aggregate UDF. See [Section 28.3.4.1, “UDF Calling Sequences for Simple Functions”](#).

For an aggregate UDF, MySQL calls the `xxx()` function after all rows in the group have been processed. You should normally never access its `UDF_ARGS` argument here but instead return a value based on your internal summary variables.

Return value handling in `xxx()` should be done the same way as for a non-aggregate UDF. See [Section 28.3.4.4, “UDF Return Values and Error Handling”](#).

The `xxx_reset()` and `xxx_add()` functions handle their `UDF_ARGS` argument the same way as functions for non-aggregate UDFs. See [Section 28.3.4.3, “UDF Argument Processing”](#).

The pointer arguments to `is_null` and `error` are the same for all calls to `xxx_reset()`, `xxx_clear()`, `xxx_add()` and `xxx()`. You can use this to remember that you got an error or whether the `xxx()` function should return `NULL`. You should not store a string into `*error`! `error` points to a single-byte variable, not to a string buffer.

`*is_null` is reset for each group (before calling `xxx_clear()`). `*error` is never reset.

If `*is_null` or `*error` are set when `xxx()` returns, MySQL returns `NULL` as the result for the group function.

28.3.4.3. UDF Argument Processing

The `args` parameter points to a `UDF_ARGS` structure that has the members listed here:

- `unsigned int arg_count`

The number of arguments. Check this value in the initialization function if you require your function to be called with a particular number of arguments. For example:

```
if (args->arg_count != 2)
{
    strcpy(message, "xxx() requires two arguments");
    return 1;
}
```

For other `UDF_ARGS` member values that are arrays, array references are zero-based. That is, refer to array members using index

values from 0 to `args->arg_count - 1`.

- `enum Item_result *arg_type`

A pointer to an array containing the types for each argument. The possible type values are `STRING_RESULT`, `INT_RESULT`, `REAL_RESULT`, and `DECIMAL_RESULT`.

To make sure that arguments are of a given type and return an error if they are not, check the `arg_type` array in the initialization function. For example:

```
if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message, "XXX() requires a string and an integer");
    return 1;
}
```

Arguments of type `DECIMAL_RESULT` are passed as strings, so you should handle them the same way as `STRING_RESULT` values.

As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the `arg_type` elements to the types you want. This causes MySQL to coerce arguments to those types for each call to `xxx()`. For example, to specify that the first two arguments should be coerced to string and integer, respectively, do this in `xxx_init()`:

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

Exact-value decimal arguments such as `1.3` or `DECIMAL` column values are passed with a type of `DECIMAL_RESULT`. However, the values are passed as strings. If you want to receive a number, use the initialization function to specify that the argument should be coerced to a `REAL_RESULT` value:

```
args->arg_type[2] = REAL_RESULT;
```

- `char **args`

`args->args` communicates information to the initialization function about the general nature of the arguments passed to your function. For a constant argument `i`, `args->args[i]` points to the argument value. (See below for instructions on how to access the value properly.) For a non-constant argument, `args->args[i]` is `0`. A constant argument is an expression that uses only constants, such as `3` or `4*7-2` or `SIN(3.14)`. A non-constant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with non-constant arguments.

For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed.

If argument `i` represents `NULL`, `args->args[i]` is a null pointer (`0`). If the argument is not `NULL`, functions can refer to it as follows:

- An argument of type `STRING_RESULT` is given as a string pointer plus a length, to allow handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. Do not assume that the string is null-terminated.
- For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a `long long` value:

```
long long int_val;
int_val = *((long long*) args->args[i]);
```

- For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a `double` value:

```
double real_val;
real_val = *((double*) args->args[i]);
```

- For an argument of type `DECIMAL_RESULT`, the value is passed as a string and should be handled like a `STRING_RESULT` value.
- `ROW_RESULT` arguments are not implemented.

- `unsigned long *lengths`

For the initialization function, the `lengths` array indicates the maximum string length for each argument. You should not change these. For each invocation of the main function, `lengths` contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialization function).

- `char *maybe_null`

For the initialization function, the `maybe_null` array indicates for each argument whether the argument value might be null (0 if no, 1 if yes).

- `char **attributes`

`args->attributes` communicates information about the names of the UDF arguments. For argument `i`, the attribute name is available as a string in `args->attributes[i]` and the attribute length is `args->attribute_lengths[i]`. Do not assume that the string is null-terminated.

By default, the name of a UDF argument is the text of the expression used to specify the argument. For UDFs, an argument may also have an optional `[AS] alias_name` clause, in which case the argument name is `alias_name`. The `attributes` value for each argument thus depends on whether an alias was given.

Suppose that a UDF `my_udf()` is invoked as follows:

```
SELECT my_udf(expr1, expr2 AS alias1, expr3 alias2);
```

In this case, the `attributes` and `attribute_lengths` arrays will have these values:

```
args->attributes[0] = "expr1"
args->attribute_lengths[0] = 5

args->attributes[1] = "alias1"
args->attribute_lengths[1] = 6

args->attributes[2] = "alias2"
args->attribute_lengths[2] = 6
```

- `unsigned long *attribute_lengths`

The `attribute_lengths` array indicates the length of each argument name.

28.3.4.4. UDF Return Values and Error Handling

The initialization function should return `0` if no error occurred and `1` otherwise. If an error occurs, `xxx_init()` should store a null-terminated error message in the `message` parameter. The message is returned to the client. The message buffer is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long long` and `double` functions. A string function should return a pointer to the result and set `*result` and `*length` to the contents and length of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

The `result` buffer that is passed to the `xxx()` function is 255 bytes long. If your result fits in this, you don't have to worry about memory allocation for results.

If your string function needs to return a string longer than 255 bytes, you must allocate the space for it with `malloc()` in your `xxx_init()` function or your `xxx()` function and free it in your `xxx_deinit()` function. You can store the allocated memory in the `ptr` slot in the `UDF_INIT` structure for reuse by future `xxx()` calls. See [Section 28.3.4.1, “UDF Calling Sequences for Simple Functions”](#).

To indicate a return value of `NULL` in the main function, set `*is_null` to `1`:

```
*is_null = 1;
```


To indicate an error return in the main function, set `*error` to 1:

```
*error = 1;
```

If `xxx()` sets `*error` to 1 for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` is not even called for subsequent rows.)

28.3.4.5. Compiling and Installing User-Defined Functions

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file `sql/udf_example.c` that is included in the MySQL source distribution.

The immediately following instructions are for Unix. Instructions for Windows are given later in this section.

The `udf_example.c` file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it's more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `sequence([const int])` returns a sequence starting from the given number or 1 if no number has been given.
- `lookup()` returns the IP number for a hostname.
- `reverse_lookup()` returns the hostname for an IP number. The function may be called either with a single string argument of the form `'xxx.xxx.xxx.xxx'` or with four numbers.

A dynamically loadable file should be compiled as a sharable object file, using a command something like this:

```
shell> gcc -shared -o udf_example.so udf_example.c
```

If you are using `gcc` with `configure` and `libtool` (which is how MySQL is configured), you should be able to create `udf_example.so` with a simpler command:

```
shell> make udf_example.la
```

After you compile a shared object containing UDFs, you must install it and tell MySQL about it. Compiling a shared object from `udf_example.c` using `gcc` directly produces a file named `udf_example.so`. Compiling the shared object using `make` produces a file named something like `udf_example.so.0.0.0` in the `.libs` directory (the exact name may vary from platform to platform). Copy the shared object to the server's plugin directory and name it `udf_example.so`. This directory is given by the value of the `plugin_dir` system variable. (**Note:** This is a change in MySQL 5.1. For earlier versions of MySQL, the shared object can be located in any directory that is searched by your system's dynamic linker.)

On some systems, the `ldconfig` program that configures the dynamic linker does not recognize a shared object unless its name begins with `lib`. In this case you should rename a file such as `udf_example.so` to `libudf_example.so`.

On Windows, you can compile user-defined functions by using the following procedure:

1. You need to obtain the BitKeeper source repository for MySQL 5.1. See [Section 2.9.3, "Installing from the Development Source Tree"](#).
2. You must obtain the CMake build utility from <http://www.cmake.org>. (Version 2.4.2 or later is required).
3. In the source repository, look in the `sql` directory. There are files named `udf_example.def` and `udf_example.c` there. Copy both files from this directory to your working directory.
4. Create a CMake `makefile` with these contents:

```
PROJECT(udf_example)

# Path for MySQL include directory
INCLUDE_DIRECTORIES("c:/mysql/include")

ADD_DEFINITIONS("-DHAVE_DLOPEN")
ADD_LIBRARY(udf_example MODULE udf_example.c udf_example.def)
TARGET_LINK_LIBRARIES(udf_example wssock32)
```

5. Create the VC project and solution files:

```
cmake -G "<Generator>"
```

Invoking `cmake --help` shows you a list of valid Generators.

6. Create `udf_example.dll`:

```
devenv udf_example.sln /build Release
```

After the shared object file has been installed, notify `mysqld` about the new functions with these statements:

```
mysql> CREATE FUNCTION metaphor RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION reverse_lookup
-> RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE AGGREGATE FUNCTION avgcost
-> RETURNS REAL SONAME 'udf_example.so';
```

Functions can be deleted using `DROP FUNCTION`:

```
mysql> DROP FUNCTION metaphor;
mysql> DROP FUNCTION myfunc_double;
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION lookup;
mysql> DROP FUNCTION reverse_lookup;
mysql> DROP FUNCTION avgcost;
```

The `CREATE FUNCTION` and `DROP FUNCTION` statements update the `func` system table in the `mysql` database. The function's name, type and shared library name are saved in the table. You must have the `INSERT` and `DELETE` privileges for the `mysql` database to create and drop functions.

You should not use `CREATE FUNCTION` to add a function that has previously been created. If you need to reinstall a function, you should remove it with `DROP FUNCTION` and then reinstall it with `CREATE FUNCTION`. You would need to do this, for example, if you recompile a new version of your function, so that `mysqld` gets the new version. Otherwise, the server continues to use the old version.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable.

If the new function will be referred to in statements that will be replicated to slave servers, you must ensure that every slave server also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

28.3.4.6. User-Defined Function Security Precautions

MySQL takes the following measures to prevent misuse of user-defined functions.

You must have the `INSERT` privilege to be able to use `CREATE FUNCTION` and the `DELETE` privilege to be able to use `DROP FUNCTION`. This is necessary because these statements add and delete rows from the `mysql.func` table.

UDFs should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. `mysqld` also supports an `--allow-suspicious-udfs` option that controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off, to prevent attempts at loading functions from shared object files other than those containing legit-

imate UDFs. If you have older UDFs that contain only the `xxx` symbol and that cannot be recompiled to include an auxiliary symbol, it may be necessary to specify the `--allow-suspicious-udfs` option. Otherwise, you should avoid enabling this capability.

UDF object files cannot be placed in arbitrary directories. They must be located in the server's plugin directory. This directory is given by the value of the `plugin_dir` system variable. (**Note:** This a change in MySQL 5.1. For earlier versions of MySQL, the shared object can be located in any directory that is searched by your system's dynamic linker.)

28.3.5. Adding a New Native Function

The procedure for adding a new native function is described here. Note that you cannot add native functions to a binary distribution because the procedure involves modifying MySQL source code. You must compile MySQL yourself from a source distribution. Also note that if you migrate to another version of MySQL (for example, when a new version is released), you need to repeat the procedure with the new version.

To add a new native MySQL function, follow these steps:

1. Add one line to `lex.h` that defines the function name in the `sql_functions[]` array.
2. If the function prototype is simple (just takes zero, one, two or three arguments), you should in `lex.h` specify `SYM(FUNC_ARGN)` (where *N* is the number of arguments) as the second argument in the `sql_functions[]` array and add a function that creates a function object in `item_create.cc`. Take a look at "ABS" and `create_funcs_abs()` for an example of this.

If the function prototype is complicated (for example, if it takes a variable number of arguments), you should add two lines to `sql_yacc.yy`. One indicates the preprocessor symbol that `yacc` should define (this should be added at the beginning of the file). Then define the function parameters and add an "item" with these parameters to the `simple_expr` parsing rule. For an example, check all occurrences of `ATAN` in `sql_yacc.yy` to see how this is done.

3. In `item_func.h`, declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.
4. In `item_func.cc`, add one of the following declarations, depending on whether you are defining a numeric or string function:

```
double Item_func_newname::val()
longlong Item_func_newname::val_int()
String *Item_func_newname::Str(String *str)
```

If you inherit your object from any of the standard items (like `Item_num_func`), you probably only have to define one of these functions and let the parent object take care of the other functions. For example, the `Item_str_func` class defines a `val()` function that executes `atof()` on the value returned by `::str()`.

5. If the function is non-deterministic, you should include the following statement in the item constructor to indicate that function results should not be cached:

```
current_thd->lex->safe_to_cache_query=0;
```

A function is non-deterministic if, given fixed values for its arguments, it can return different results for different invocations.

6. You should probably also define the following object function:

```
void Item_func_newname::fix_length_and_dec()
```

This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a `NULL` value. The function can check whether any of the function arguments can return `NULL` by checking the arguments' `maybe_null` variable. You can take a look at `Item_func_mod::fix_length_and_dec` for a typical example of how to do this.

All functions must be thread-safe. In other words, don't use any global or static variables in the functions without protecting them with mutexes)

If you want to return `NULL`, from `::val()`, `::val_int()` or `::str()` you should set `null_value` to 1 and return 0.

For `::str()` object functions, there are some additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result. (For more information about the `String` type, take a look at the `sql_string.h` file.)
- The `::str()` function should return the string that holds the result or `(char*) 0` if the result is `NULL`.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

If the new native function will be referred to in statements that will be replicated to slave servers, you must ensure that every slave server also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

28.4. Adding New Procedures to MySQL

In MySQL, you can define a procedure in C++ that can access and modify the data in a query before it is sent to the client. The modification can be done on a row-by-row or `GROUP BY` level.

We have created an example procedure to show you what can be done.

Additionally, we recommend that you take a look at [mylua](#). With this you can use the LUA language to load a procedure at runtime into `mysqld`.

28.4.1. Procedure Analyse

`analyse([max_elements],[max_memory])`

This procedure is defined in the `sql/sql_analyse.cc` file. It examines the result from a query and returns an analysis of the results that suggests optimal data types for each column. To obtain this analysis, append `PROCEDURE ANALYSE` to the end of a `SELECT` statement:

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements],[max_memory])
```

For example:

```
SELECT col1, col2 FROM table1 PROCEDURE ANALYSE(10, 2000);
```

The results show some statistics for the values returned by the query, and propose an optimal data type for the columns. This can be helpful for checking your existing tables, or after importing new data. You may need to try different settings for the arguments so that `PROCEDURE ANALYSE()` does not suggest the `ENUM` data type when it is not appropriate.

The arguments are optional and are used as follows:

- `max_elements` (default 256) is the maximum number of distinct values that `analyse` notices per column. This is used by `analyse` to check whether the optimal data type should be of type `ENUM`.
- `max_memory` (default 8192) is the maximum amount of memory that `analyse` should allocate per column while trying to find all distinct values.

28.4.2. Writing a Procedure

For the moment, the only documentation for this is the source.

You can find all information about procedures by examining the following files:

- `sql/sql_analyse.cc`
- `sql/procedure.h`
- `sql/procedure.cc`
- `sql/sql_select.cc`

Appendix A. MySQL 5.1 Frequently Asked Questions

A.1. MySQL 5.1 FAQ — General

Questions

- [30.1.1](#): When did MySQL 5.1 become production-ready (GA)?
- [30.1.2](#): Can MySQL 5.1 do subqueries?
- [30.1.3](#): Can MySQL 5.1 perform multiple-table inserts, updates, and deletes?
- [30.1.4](#): Does MySQL 5.1 have a Query Cache? Does it work on Server, Instance or Database?
- [30.1.5](#): Does MySQL 5.1 have Sequences?
- [30.1.6](#): Does MySQL 5.1 have a `NOW()` function with fractions of seconds?
- [30.1.7](#): Does MySQL 5.1 work with multi-core processors?
- [30.1.8](#): Is there a hot backup tool for MyISAM like InnoDB Hot Backup?
- [30.1.9](#): Have there been any improvements in error reporting when foreign keys fail? Does MySQL now report which column and reference failed?
- [30.1.10](#): Can MySQL 5.1 perform ACID transactions?

Questions and Answers

30.1.1: When did MySQL 5.1 become production-ready (GA)?

MySQL 5.0.15 was released for production use on 19 October 2005. We are now working on MySQL 5.1, which is currently in beta.

30.1.2: Can MySQL 5.1 do subqueries?

Yes. See [Section 12.2.8](#), “Subquery Syntax”.

30.1.3: Can MySQL 5.1 perform multiple-table inserts, updates, and deletes?

Yes. For the syntax required to perform multiple-table updates, see [Section 12.2.10](#), “UPDATE Syntax”; for that required to perform multiple-table deletes, see [Section 12.2.1](#), “DELETE Syntax”.

A multiple-table insert can be accomplished using a trigger whose `FOR EACH ROW` clause contains multiple `INSERT` statements within a `BEGIN . . . END` block. See [Section 20.3](#), “Using Triggers”.

30.1.4: Does MySQL 5.1 have a Query Cache? Does it work on Server, Instance or Database?

Yes. The query cache operates on the server level, caching complete result sets matched with the original query string. If an exactly identical query is made (which often happens, particularly in web applications), no parsing or execution is necessary; the result is sent directly from the cache. Various tuning options are available. See [Section 6.5.4](#), “The MySQL Query Cache”.

30.1.5: Does MySQL 5.1 have Sequences?

No. However, MySQL has an `AUTO_INCREMENT` system, which in MySQL 5.1 can also handle inserts in a multi-master replication setup. With the `--auto-increment-increment` and `--auto-increment-offset` startup options, you can set each server to generate auto-increment values that don't conflict with other servers. The `--auto-increment-increment` value should be greater than the number of servers, and each server should have a unique offset.

30.1.6: Does MySQL 5.1 have a `NOW()` function with fractions of seconds?

No. This is on the MySQL roadmap as a “rolling feature”. This means that it is not a flagship feature, but will be implemented, development time permitting. Specific customer demand may change this scheduling.

However, MySQL does parse time strings with a fractional component. See [Section 10.3.2, “The TIME Type”](#).

30.1.7: Does MySQL 5.1 work with multi-core processors?

Yes. MySQL is fully multi-threaded, and will make use of multiple CPUs, provided that the operating system supports them.

30.1.8: Is there a hot backup tool for MyISAM like InnoDB Hot Backup?

This is currently under development for a future MySQL release.

30.1.9: Have there been there any improvements in error reporting when foreign keys fail? Does MySQL now report which column and reference failed?

The foreign key support in [InnoDB](#) has seen improvements in each major version of MySQL. Foreign key support generic to all storage engines is scheduled for MySQL 5.2; this should resolve any inadequacies in the current storage engine specific implementation.

30.1.10: Can MySQL 5.1 perform ACID transactions?

Yes. All current MySQL versions support transactions. The [InnoDB](#) storage engine offers full ACID transactions with row-level locking, multi-versioning, non-locking repeatable reads, and all four SQL standard isolation levels.

The [NDB](#) storage engine supports the [READ COMMITTED](#) transaction isolation level only.

A.2. MySQL 5.1 FAQ — Storage Engines

Questions

- [30.2.1](#): Where can I obtain complete documentation for MySQL storage engines and the pluggable storage engine architecture?
- [30.2.2](#): Are there any new storage engines in MySQL 5.1?
- [30.2.3](#): Have any storage engines been removed in MySQL 5.1?
- [30.2.4](#): What are the unique benefits of the [ARCHIVE](#) storage engine?
- [30.2.5](#): Do the new features in MySQL 5.1 apply to all storage engines?

Questions and Answers

30.2.1: Where can I obtain complete documentation for MySQL storage engines and the pluggable storage engine architecture?

See [Chapter 13, Storage Engines](#). That chapter contains information about all MySQL storage engines except for the [NDB](#) storage engine used for MySQL Cluster; [NDB](#) is covered in [Chapter 16, MySQL Cluster](#).

MySQL Enterprise

For expert advice about the storage engine(s) most suitable to your circumstances subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

30.2.2: Are there any new storage engines in MySQL 5.1?

MySQL 5.1 introduces an alpha version of the new Falcon storage engine.

Falcon support is not available in the standard MySQL 5.1 release. Falcon is available only in a specially forked release of MySQL 5.1. Information is provided here for evaluation purposes only.

For information about the Falcon storage engine, see [Using Falcon](#).

There have also been significant improvements in existing storage engines, in particular for the [NDB](#) storage engine that forms the basis for MySQL Cluster.

30.2.3: Have any storage engines been removed in MySQL 5.1?

Yes. MySQL 5.1 no longer supports the [BDB](#) storage engine. Any existing [BDB](#) tables should be converted to another storage engine before upgrading to MySQL 5.1.

30.2.4: What are the unique benefits of the [ARCHIVE](#) storage engine?

The [ARCHIVE](#) storage engine is ideally suited for storing large amounts of data without indexes; it has a very small footprint, and performs selects using table scans. See [Section 13.10, “The ARCHIVE Storage Engine”](#), for details.

30.2.5: Do the new features in MySQL 5.1 apply to all storage engines?

The general new features such as views, stored procedures, triggers, [INFORMATION_SCHEMA](#), precision math ([DECIMAL](#) column type), and the [BIT](#) column type, apply to all storage engines. There are also additions and changes for specific storage engines.

A.3. MySQL 5.1 FAQ — Server SQL Mode

Questions

- [30.3.1](#): What are server SQL modes?
- [30.3.2](#): How many server SQL modes are there?
- [30.3.3](#): How do you determine the server SQL mode?
- [30.3.4](#): Is the mode dependent on the database or connection?
- [30.3.5](#): Can the rules for strict mode be extended?
- [30.3.6](#): Does strict mode impact performance?
- [30.3.7](#): What is the default server SQL mode when MySQL 5.1 is installed?

Questions and Answers

30.3.1: What are server SQL modes?

Server SQL modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers. The MySQL Server applies these modes individually to different clients. For more information, see [Section 5.2.6, “SQL Modes”](#).

30.3.2: How many server SQL modes are there?

Each mode can be independently switched on and off. See [Section 5.2.6, “SQL Modes”](#), for a complete list of available modes.

30.3.3: How do you determine the server SQL mode?

You can set the default SQL mode (for `mysqld` startup) with the `--sql-mode` option. Using the statement `SET [SESSION|GLOBAL] sql_mode='modes'`, you can change the settings from within a connection, either locally to the connection, or to take effect globally. You can retrieve the current mode by issuing a `SELECT @@sql_mode` statement.

30.3.4: Is the mode dependent on the database or connection?

A mode is not linked to a particular database. Modes can be set locally to the session (connection), or globally for the server. You can change these settings using `SET [SESSION|GLOBAL] sql_mode='modes'`.

30.3.5: Can the rules for strict mode be extended?

When we refer to *strict mode*, we mean a mode where at least one of the modes [TRADITIONAL](#), [STRICT_TRANS_TABLES](#), or [STRICT_ALL_TABLES](#) is enabled. Options can be combined, so you can add additional restrictions to a mode. See [Section 5.2.6, “SQL Modes”](#), for more information.

30.3.6: Does strict mode impact performance?

The intensive validation of input data that some settings requires more time than if the validation is not done. While the performance impact is not that great, if you do not require such validation (perhaps your application already handles all of this), then MySQL gives you the option of leaving strict mode disabled. However — if you do require it — strict mode can provide such validation.

30.3.7: What is the default server SQL mode when MySQL 5.1 is installed?

By default, no special modes are enabled. See [Section 5.2.6, “SQL Modes”](#), for information about all available modes and MySQL's default behavior.

A.4. MySQL 5.1 FAQ — Stored Procedures

Questions

- [30.4.1](#): Does MySQL 5.1 support stored procedures?
- [30.4.2](#): Where can I find documentation for MySQL stored procedures and stored functions?
- [30.4.3](#): Is there a discussion forum for MySQL stored procedures?
- [30.4.4](#): Where can I find the ANSI SQL 2003 specification for stored procedures?
- [30.4.5](#): How do you manage stored routines?
- [30.4.6](#): Is there a way to view all stored procedures and stored functions in a given database?
- [30.4.7](#): Where are stored procedures stored?
- [30.4.8](#): Is it possible to group stored procedures or stored functions into packages?
- [30.4.9](#): Can a stored procedure call another stored procedure?
- [30.4.10](#): Can a stored procedure call a trigger?
- [30.4.11](#): Can a stored procedure access tables?
- [30.4.12](#): Do stored procedures have a statement for raising application errors?
- [30.4.13](#): Do stored procedures provide exception handling?
- [30.4.14](#): Can MySQL 5.1 stored routines return result sets?
- [30.4.15](#): Is [WITH RECOMPILE](#) supported for stored procedures?
- [30.4.16](#): Is there a MySQL equivalent to using [mod_plsql](#) as a gateway on Apache to talk directly to a stored procedure in the database?
- [30.4.17](#): Can I pass an array as input to a stored procedure?
- [30.4.18](#): Can I pass a cursor as an [IN](#) parameter to a stored procedure?
- [30.4.19](#): Can I return a cursor as an [OUT](#) parameter from a stored procedure?
- [30.4.20](#): Can I print out a variable's value within a stored routine for debugging purposes?
- [30.4.21](#): Can I commit or roll back transactions inside a stored procedure?
- [30.4.22](#): Do MySQL 5.1 stored procedures and functions work with replication?
- [30.4.23](#): Are stored procedures and functions created on a master server replicated to a slave?
- [30.4.24](#): How are actions that take place inside stored procedures and functions replicated?
- [30.4.25](#): Are there special security requirements for using stored procedures and functions together with replication?
- [30.4.26](#): What limitations exist for replicating stored procedure and function actions?
- [30.4.27](#): Do the preceding limitations affect MySQL's ability to do point-in-time recovery?
- [30.4.28](#): What is being done to correct the aforementioned limitations?

Questions and Answers

30.4.1: Does MySQL 5.1 support stored procedures?

Yes. MySQL 5.1 supports two types of stored routines — stored procedures and stored functions.

30.4.2: Where can I find documentation for MySQL stored procedures and stored functions?

See [Chapter 19, *Stored Procedures and Functions*](#).

30.4.3: Is there a discussion forum for MySQL stored procedures?

Yes. See <http://forums.mysql.com/list.php?98>.

30.4.4: Where can I find the ANSI SQL 2003 specification for stored procedures?

Unfortunately, the official specifications are not freely available (ANSI makes them available for purchase). However, there are books — such as *SQL-99 Complete, Really* by Peter Gultzan and Trudy Pelzer — which give a comprehensive overview of the standard, including coverage of stored procedures.

30.4.5: How do you manage stored routines?

It is always good practice to use a clear naming scheme for your stored routines. You can manage stored procedures with `CREATE [FUNCTION|PROCEDURE]`, `ALTER [FUNCTION|PROCEDURE]`, `DROP [FUNCTION|PROCEDURE]`, and `SHOW CREATE [FUNCTION|PROCEDURE]`. You can obtain information about existing stored procedures using the `ROUTINES` table in the `INFORMATION_SCHEMA` database (see [Section 23.14, “The INFORMATION_SCHEMA ROUTINES Table”](#)).

30.4.6: Is there a way to view all stored procedures and stored functions in a given database?

Yes. For a database named `dbname`, use this query on the `INFORMATION_SCHEMA.ROUTINES` table:

```
SELECT ROUTINE_TYPE, ROUTINE_NAME
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_SCHEMA='dbname';
```

For more information, see [Section 23.14, “The INFORMATION_SCHEMA ROUTINES Table”](#).

The body of a stored routine can be viewed using `SHOW CREATE FUNCTION` (for a stored function) or `SHOW CREATE PROCEDURE` (for a stored procedure). See [Section 12.5.4.8, “SHOW CREATE PROCEDURE and SHOW CREATE FUNCTION Syntax”](#), for more information.

30.4.7: Where are stored procedures stored?

In the `proc` table of the `mysql` system database. However, you should not access the tables in the system database directly. Instead, use `SHOW CREATE FUNCTION` to obtain information about stored functions, and `SHOW CREATE PROCEDURE` to obtain information about stored procedures. See [Section 12.5.4.8, “SHOW CREATE PROCEDURE and SHOW CREATE FUNCTION Syntax”](#), for more information about these statements.

You can also query the `ROUTINES` table in the `INFORMATION_SCHEMA` database — see [Section 23.14, “The INFORMATION_SCHEMA ROUTINES Table”](#), for information about this table.

30.4.8: Is it possible to group stored procedures or stored functions into packages?

No. This is not supported in MySQL 5.1.

30.4.9: Can a stored procedure call another stored procedure?

Yes.

30.4.10: Can a stored procedure call a trigger?

A stored procedure can execute an SQL statement, such as an `UPDATE`, that causes a trigger to fire.

30.4.11: Can a stored procedure access tables?

Yes. A stored procedure can access one or more tables as required.

30.4.12: Do stored procedures have a statement for raising application errors?

Not in MySQL 5.1. We intend to implement the SQL standard [SIGNAL](#) and [RESIGNAL](#) statements in a future MySQL release.

30.4.13: Do stored procedures provide exception handling?

MySQL implements [HANDLER](#) definitions according to the SQL standard. See [Section 19.2.8.2, “DECLARE Handlers”](#), for details.

30.4.14: Can MySQL 5.1 stored routines return result sets?

Stored procedures can, but stored functions cannot. If you perform an ordinary [SELECT](#) inside a stored procedure, the result set is returned directly to the client. You need to use the MySQL 4.1 (or above) client-server protocol for this to work. This means that — for instance — in PHP, you need to use the [mysqli](#) extension rather than the old [mysql](#) extension.

30.4.15: Is [WITH RECOMPILE](#) supported for stored procedures?

Not in MySQL 5.1.

30.4.16: Is there a MySQL equivalent to using [mod_php](#) as a gateway on Apache to talk directly to a stored procedure in the database?

There is no equivalent in MySQL 5.1.

30.4.17: Can I pass an array as input to a stored procedure?

Not in MySQL 5.1.

30.4.18: Can I pass a cursor as an [IN](#) parameter to a stored procedure?

In MySQL 5.1, cursors are available inside stored procedures only.

30.4.19: Can I return a cursor as an [OUT](#) parameter from a stored procedure?

In MySQL 5.1, cursors are available inside stored procedures only. However, if you do not open a cursor on a [SELECT](#), the result will be sent directly to the client. You can also [SELECT INTO](#) variables. See [Section 12.2.7, “SELECT Syntax”](#).

30.4.20: Can I print out a variable's value within a stored routine for debugging purposes?

Yes, you can do this in a *stored procedure*, but not in a stored function. If you perform an ordinary [SELECT](#) inside a stored procedure, the result set is returned directly to the client. You will need to use the MySQL 4.1 (or above) client-server protocol for this to work. This means that — for instance — in PHP, you need to use the [mysqli](#) extension rather than the old [mysql](#) extension.

30.4.21: Can I commit or roll back transactions inside a stored procedure?

Yes. However, you cannot perform transactional operations within a stored function.

30.4.22: Do MySQL 5.1 stored procedures and functions work with replication?

Yes, standard actions carried out in stored procedures and functions are replicated from a master MySQL server to a slave server. There are a few limitations that are described in detail in [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

30.4.23: Are stored procedures and functions created on a master server replicated to a slave?

Yes, creation of stored procedures and functions carried out through normal DDL statements on a master server are replicated to a slave, so the objects will exist on both servers. [ALTER](#) and [DROP](#) statements for stored procedures and functions are also replicated.

30.4.24: How are actions that take place inside stored procedures and functions replicated?

MySQL records each DML event that occurs in a stored procedure and replicates those individual actions to a slave server. The actual calls made to execute stored procedures are not replicated.

Stored functions that change data are logged as function invocations, not as the DML events that occur inside each function.

30.4.25: Are there special security requirements for using stored procedures and functions together with replication?

Yes. Because a slave server has authority to execute any statement read from a master's binary log, special security constraints exist for using stored functions with replication. If replication or binary logging in general (for the purpose of point-in-time recovery) is active,

then MySQL DBAs have two security options open to them:

1. Any user wishing to create stored functions must be granted the [SUPER](#) privilege.
2. Alternatively, a DBA can set the [log_bin_trust_function_creators](#) system variable to 1, which enables anyone with the standard [CREATE ROUTINE](#) privilege to create stored functions.

30.4.26: What limitations exist for replicating stored procedure and function actions?

Non-deterministic (random) or time-based actions embedded in stored procedures may not replicate properly. By their very nature, randomly produced results are not predictable and cannot be exactly reproduced, and therefore, random actions replicated to a slave will not mirror those performed on a master. Note that declaring stored functions to be [DETERMINISTIC](#) or setting the [log_bin_trust_function_creators](#) system variable to 0 will not allow random-valued operations to be invoked.

In addition, time-based actions cannot be reproduced on a slave because the timing of such actions in a stored procedure is not reproducible through the binary log used for replication. It records only DML events and does not factor in timing constraints.

Finally, non-transactional tables for which errors occur during large DML actions (such as bulk inserts) may experience replication issues in that a master may be partially updated from DML activity, but no updates are done to the slave because of the errors that occurred. A workaround is for a function's DML actions to be carried out with the [IGNORE](#) keyword so that updates on the master that cause errors are ignored and updates that do not cause errors are replicated to the slave.

30.4.27: Do the preceding limitations affect MySQL's ability to do point-in-time recovery?

The same limitations that affect replication do affect point-in-time recovery.

30.4.28: What is being done to correct the aforementioned limitations?

As of MySQL 5.1.5, you can choose either statement-based replication or row-based replication. The original replication implementation is based on statement-based binary logging. Row-based binary logging resolves the limitations mentioned earlier.

Beginning with MySQL 5.1.8, *mixed* replication is also available (by starting the server with [--binlog-format=mixed](#)). This hybrid, “smart” form of replication “knows” whether statement-level replication can safely be used, or row-level replication is required.

For additional information, see [Section 15.1.2, “Replication Formats”](#).

A.5. MySQL 5.1 FAQ — Triggers

Questions

- [30.5.1](#): Where can I find the documentation for MySQL 5.1 triggers?
- [30.5.2](#): Is there a discussion forum for MySQL Triggers?
- [30.5.3](#): Does MySQL 5.1 have statement-level or row-level triggers?
- [30.5.4](#): Are there any default triggers?
- [30.5.5](#): How are triggers managed in MySQL?
- [30.5.6](#): Is there a way to view all triggers in a given database?
- [30.5.7](#): Where are triggers stored?
- [30.5.8](#): Can a trigger call a stored procedure?
- [30.5.9](#): Can triggers access tables?
- [30.5.10](#): Can triggers call an external application through a UDF?
- [30.5.11](#): Is possible for a trigger to update tables on a remote server?
- [30.5.12](#): Do triggers work with replication?

- [30.5.13](#): How are actions carried out through triggers on a master replicated to a slave?

Questions and Answers

30.5.1: Where can I find the documentation for MySQL 5.1 triggers?

See [Chapter 20, Triggers](#).

30.5.2: Is there a discussion forum for MySQL Triggers?

Yes. It is available at <http://forums.mysql.com/list.php?99>.

30.5.3: Does MySQL 5.1 have statement-level or row-level triggers?

In MySQL 5.1, all triggers are **FOR EACH ROW** — that is, the trigger is activated for each row that is inserted, updated, or deleted. MySQL 5.1 does not support triggers using **FOR EACH STATEMENT**.

30.5.4: Are there any default triggers?

Not explicitly. MySQL does have specific special behavior for some **TIMESTAMP** columns, as well as for columns which are defined using **AUTO_INCREMENT**.

30.5.5: How are triggers managed in MySQL?

In MySQL 5.1, triggers can be created using the **CREATE TRIGGER** statement, and dropped using **DROP TRIGGER**. See [Section 20.1, “CREATE TRIGGER Syntax”](#), and [Section 20.2, “DROP TRIGGER Syntax”](#), for more about these statements.

Information about triggers can be obtained by querying the **INFORMATION_SCHEMA.TRIGGERS** table. See [Section 23.16, “The INFORMATION_SCHEMA TRIGGERS Table”](#).

30.5.6: Is there a way to view all triggers in a given database?

Yes. You can obtain a listing of all triggers defined on database **dbname** using a query on the **INFORMATION_SCHEMA.TRIGGERS** table such as the one shown here:

```
SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_STATEMENT
FROM INFORMATION_SCHEMA.TRIGGERS
WHERE TRIGGER_SCHEMA='dbname';
```

For more information about this table, see [Section 23.16, “The INFORMATION_SCHEMA TRIGGERS Table”](#).

You can also use the **SHOW TRIGGERS** statement, which is specific to MySQL. See [Section 12.5.4.30, “SHOW TRIGGERS Syntax”](#).

30.5.7: Where are triggers stored?

Triggers are currently stored in **.TRG** files, with one such file one per table. In other words, a trigger belongs to a table.

In the future, we plan to change this so that trigger information will be included in the **.FRM** file that defines the structure of the table. We also plan to make triggers database-level objects — rather than table-level objects as they are now — to bring them into compliance with the SQL standard.

30.5.8: Can a trigger call a stored procedure?

Yes.

30.5.9: Can triggers access tables?

A trigger can access both old and new data in its own table. Through a stored procedure, or a multiple-table update or delete statement, a trigger can also affect other tables.

30.5.10: Can triggers call an external application through a UDF?

No, not at present.

30.5.11: Is possible for a trigger to update tables on a remote server?

Yes. A table on a remote server could be updated using the [FEDERATED](#) storage engine. (See [Section 13.9](#), “[The FEDERATED Storage Engine](#)”).

30.5.12: Do triggers work with replication?

Triggers and replication in MySQL 5.1 work in the same way as in most other database engines: Actions carried out through triggers on a master are not replicated to a slave server. Instead, triggers that exist on tables that reside on a MySQL master server need to be created on the corresponding tables on any MySQL slave servers so that the triggers activate on the slaves as well as the master.

30.5.13: How are actions carried out through triggers on a master replicated to a slave?

First, the triggers that exist on a master must be re-created on the slave server. Once this is done, the replication flow works as any other standard DML statement that participates in replication. For example, consider a table [EMP](#) that has an [AFTER](#) insert trigger, which exists on a master MySQL server. The same [EMP](#) table and [AFTER](#) insert trigger exist on the slave server as well. The replication flow would be:

1. An [INSERT](#) statement is made to [EMP](#).
2. The [AFTER](#) trigger on [EMP](#) activates.
3. The [INSERT](#) statement is written to the binary log.
4. The replication slave picks up the [INSERT](#) statement to [EMP](#) and executes it.
5. The [AFTER](#) trigger on [EMP](#) that exists on the slave activates.

A.6. MySQL 5.1 FAQ — Views

Questions

- [30.6.1](#): Where can I find documentation covering MySQL Views?
- [30.6.2](#): Is there a discussion forum for MySQL Views?
- [30.6.3](#): What happens to a view if an underlying table is dropped or renamed?
- [30.6.4](#): Does MySQL 5.1 have table snapshots?
- [30.6.5](#): Does MySQL 5.1 have materialized views?
- [30.6.6](#): Can you insert into views that are based on joins?

Questions and Answers

30.6.1: Where can I find documentation covering MySQL Views?

See [Chapter 22](#), [Views](#).

30.6.2: Is there a discussion forum for MySQL Views?

Yes. See <http://forums.mysql.com/list.php?100>

30.6.3: What happens to a view if an underlying table is dropped or renamed?

After a view has been created, it is possible to drop or alter a table or view to which the definition refers. To check a view definition for problems of this kind, use the [CHECK TABLE](#) statement. (See [Section 12.5.2.3](#), “[CHECK TABLE Syntax](#)”).

30.6.4: Does MySQL 5.1 have table snapshots?

No.

30.6.5: Does MySQL 5.1 have materialized views?

No.

30.6.6: Can you insert into views that are based on joins?

It is possible, provided that your `INSERT` statement has a column list that makes it clear there's only one table involved.

You *cannot* insert into multiple tables with a single insert on a view.

A.7. MySQL 5.0 FAQ — INFORMATION_SCHEMA

Questions

- [30.7.1](#): Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?
- [30.7.2](#): Is there a discussion forum for `INFORMATION_SCHEMA`?
- [30.7.3](#): Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?
- [30.7.4](#): What is the difference between the Oracle Data Dictionary and MySQL's `INFORMATION_SCHEMA`?
- [30.7.5](#): Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?

Questions and Answers

30.7.1: Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?

See [Chapter 23, The `INFORMATION\_SCHEMA` Database](#)

30.7.2: Is there a discussion forum for `INFORMATION_SCHEMA`?

See <http://forums.mysql.com/list.php?101>.

30.7.3: Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?

Unfortunately, the official specifications are not freely available. (ANSI makes them available for purchase.) However, there are books available — such as *SQL-99 Complete, Really* by Peter Gultzan and Trudy Pelzer — which give a comprehensive overview of the standard, including `INFORMATION_SCHEMA`.

30.7.4: What is the difference between the Oracle Data Dictionary and MySQL's `INFORMATION_SCHEMA`?

Both Oracle and MySQL provide metadata in tables. However, Oracle and MySQL use different table names and column names. MySQL's implementation is more similar to those found in DB2 and SQL Server, which also support `INFORMATION_SCHEMA` as defined in the SQL standard.

30.7.5: Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?

No. Since applications may rely on a certain standard structure, this should not be modified. For this reason, *MySQL AB cannot support bugs or other issues which result from modifying `INFORMATION_SCHEMA` tables or data.*

A.8. MySQL 5.1 FAQ — Migration

Questions

- [30.8.1](#): Where can I find information on how to migrate from MySQL 5.0 to MySQL 5.1?
- [30.8.2](#): How has storage engine (table type) support changed in MySQL 5.1 from previous versions?

Questions and Answers

30.8.1: Where can I find information on how to migrate from MySQL 5.0 to MySQL 5.1?

For detailed upgrade information, see [Section 2.11, “Upgrading MySQL”](#). We recommend that you do not skip a major version when upgrading, but rather complete the process in steps, upgrading from one major version to the next in each step. This may seem more complicated, but it will save you time and trouble — if you encounter problems during the upgrade, their origin will be easier to identify, either by you or — if you have a MySQL Network subscription — by MySQL support.

30.8.2: How has storage engine (table type) support changed in MySQL 5.1 from previous versions?

Storage engine support has changed as follows:

- Support for [ISAM](#) tables was removed in MySQL 5.0 and you should now use the [MyISAM](#) storage engine in place of [ISAM](#). To convert a table *tblname* from [ISAM](#) to [MyISAM](#), simply issue a statement such as this one:

```
ALTER TABLE tblname ENGINE=MYISAM;
```

- Internal [RAID](#) for [MyISAM](#) tables was also removed in MySQL 5.0. This was formerly used to allow large tables in filesystems that did not support file sizes greater than 2GB. All modern filesystems allow for larger tables; in addition, there are now other solutions such as [MERGE](#) tables and views.
- The [VARCHAR](#) column type now retains trailing spaces in all storage engines.
- [MEMORY](#) tables (formerly known as [HEAP](#) tables) can also contain [VARCHAR](#) columns.

A.9. MySQL 5.1 FAQ — Security

Questions

- [30.9.1](#): Where can I find documentation that addresses security issues for MySQL?
- [30.9.2](#): Does MySQL 5.1 have native support for SSL?
- [30.9.3](#): Is SSL support built into MySQL binaries, or must I recompile the binary myself to enable it?
- [30.9.4](#): Does MySQL 5.1 have built-in authentication against LDAP directories?
- [30.9.5](#): Does MySQL 5.1 include support for Roles Based Access Control (RBAC)?

Questions and Answers

30.9.1: Where can I find documentation that addresses security issues for MySQL?

The best place to start is [Section 5.6, “General Security Issues”](#).

Other portions of the MySQL Documentation which you may find useful with regard to specific security concerns include the following:

- [Section 5.6.1, “General Security Guidelines”](#).
- [Section 5.6.2, “Making MySQL Secure Against Attackers”](#).
- [Section B.1.4.1, “How to Reset the Root Password”](#).
- [Section 5.6.5, “How to Run MySQL as a Normal User”](#).
- [Section 28.3.4.6, “User-Defined Function Security Precautions”](#).
- [Section 5.6.3, “Security-Related `mysqld` Options”](#).
- [Section 5.6.4, “Security Issues with `LOAD DATA LOCAL`”](#).
- [Section 2.10, “Post-Installation Setup and Testing”](#).

- [Section 2.13.1.11, “SELinux Notes”](#).
- [Section 5.8.7.1, “Basic SSL Concepts”](#).

MySQL Enterprise

The MySQL Enterprise Monitor enforces best practices for maximizing the security of your servers. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

30.9.2: Does MySQL 5.1 have native support for SSL?

Most 5.1 binaries have support for SSL connections between the client and server. We can't currently build with the new YaSSL library everywhere, as it's still quite new and does not compile on all platforms yet. See [Section 5.8.7, “Using Secure Connections”](#).

You can also tunnel a connection via SSH, if (for instance) if the client application doesn't support SSL connections. For an example, see [Section 5.8.7.5, “Connecting to MySQL Remotely from Windows with SSH”](#).

30.9.3: Is SSL support be built into MySQL binaries, or must I recompile the binary myself to enable it?

Most 5.1 binaries have SSL enabled for client-server connections that are secured, authenticated, or both. However, the YaSSL library currently does not compile on all platforms. See [Section 5.8.7, “Using Secure Connections”](#), for a complete listing of supported and unsupported platforms.

30.9.4: Does MySQL 5.1 have built-in authentication against LDAP directories?

No. Support for external authentication methods is on the MySQL roadmap as a “rolling feature”, which means that we plan to implement it in the future, but we have not yet determined when this will be done.

30.9.5: Does MySQL 5.1 include support for Roles Based Access Control (RBAC)?

No. Support for roles is on the MySQL roadmap as a “rolling feature”, which means that we plan to implement it in the future, but we have not yet determined when this will be done.

A.10. MySQL 5.1 FAQ — MySQL Cluster

In the following section, we provide answers to questions that are most frequently asked about MySQL Cluster and the [NDB](#) storage engine.

Questions

- [30.10.1](#): What does “NDB” mean?
- [30.10.2](#): What's the difference in using Cluster vs using replication?
- [30.10.3](#): Do I need to do any special networking to run Cluster? How do computers in a cluster communicate?
- [30.10.4](#): How many computers do I need to run a cluster, and why?
- [30.10.5](#): What do the different computers do in a MySQL Cluster?
- [30.10.6](#): With which operating systems can I use Cluster?
- [30.10.7](#): What are the hardware requirements for running MySQL Cluster?
- [30.10.8](#): How much RAM do I need? Is it possible to use disk memory at all?
- [30.10.9](#): What filesystems can I use with MySQL Cluster? What about network filesystems or network shares?
- [30.10.10](#): Can I run MySQL Cluster nodes inside virtual machines (such as those created by VMWare, Parallels, or Xen)?
- [30.10.11](#): I'm trying to populate a Cluster database. The loading process terminates prematurely and I get an error message like this one:

```
ERROR 1114: THE TABLE 'MY_CLUSTER_TABLE' IS FULL
```


Why is this happening?

- [30.10.12](#): MySQL Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?
- [30.10.13](#): Do I have to learn a new programming or query language to use MySQL Cluster?
- [30.10.14](#): How do I find out what an error or warning message means when using MySQL Cluster?
- [30.10.15](#): Is MySQL Cluster transaction-safe? What isolation levels are supported?
- [30.10.16](#): What storage engines are supported by MySQL Cluster?
- [30.10.17](#): Which versions of the MySQL software support Cluster? Do I have to compile from source?
- [30.10.18](#): In the event of a catastrophic failure — say, for instance, the whole city loses power *and* my UPS fails — would I lose all my data?
- [30.10.19](#): Is it possible to use [FULLTEXT](#) indexes with Cluster?
- [30.10.20](#): Can I run multiple nodes on a single computer?
- [30.10.21](#): Can I add nodes to a cluster without restarting it?
- [30.10.22](#): Are there any limitations that I should be aware of when using MySQL Cluster?
- [30.10.23](#): How do I import an existing MySQL database into a cluster?
- [30.10.24](#): How do cluster nodes communicate with one another?
- [30.10.25](#): What is an *arbitrator*?
- [30.10.26](#): What data types are supported by MySQL Cluster?
- [30.10.27](#): How do I start and stop MySQL Cluster?
- [30.10.28](#): What happens to cluster data when the cluster is shut down?
- [30.10.29](#): Is it helpful to have more than one management node for a cluster?
- [30.10.30](#): Can I mix different kinds of hardware and operating systems in one MySQL Cluster?
- [30.10.31](#): Can I run two data nodes on a single host? Two SQL nodes?
- [30.10.32](#): Can I use hostnames with MySQL Cluster?
- [30.10.33](#): How do I handle MySQL users in a Cluster having multiple MySQL servers?

Questions and Answers

30.10.1: What does “NDB” mean?

This stands for “Network Database”. [NDB](#) (also known as [NDB Cluster](#) or [NDBCLUSTER](#)) is the storage engine that enables clustering in MySQL.

30.10.2: What's the difference in using Cluster vs using replication?

In a replication setup, a master MySQL server updates one or more slaves. Transactions are committed sequentially, and a slow transaction can cause the slave to lag behind the master. This means that if the master fails, it is possible that the slave might not have recorded the last few transactions. If a transaction-safe engine such as [InnoDB](#) is being used, a transaction will either be complete on the slave or not applied at all, but replication does not guarantee that all data on the master and the slave will be consistent at all times. In MySQL Cluster, all data nodes are kept in synchrony, and a transaction committed by any one data node is committed for all data nodes. In the event of a data node failure, all remaining data nodes remain in a consistent state.

In short, whereas standard MySQL replication is asynchronous, MySQL Cluster is synchronous.

We have implemented (asynchronous) replication for Cluster in MySQL 5.1. This includes the capability to replicate both between two clusters, and from a MySQL cluster to a non-Cluster MySQL server. See [Section 16.10, “MySQL Cluster Replication”](#).

30.10.3: Do I need to do any special networking to run Cluster? How do computers in a cluster communicate?

MySQL Cluster is intended to be used in a high-bandwidth environment, with computers connecting via TCP/IP. Its performance depends directly upon the connection speed between the cluster's computers. The minimum connectivity requirements for Cluster include a typical 100-megabit Ethernet network or the equivalent. We recommend you use gigabit Ethernet whenever available.

The faster SCI protocol is also supported, but requires special hardware. See [Section 16.12, “Using High-Speed Interconnects with MySQL Cluster”](#), for more information about SCI.

30.10.4: How many computers do I need to run a cluster, and why?

A minimum of three computers is required to run a viable cluster. However, the minimum **recommended** number of computers in a MySQL Cluster is four: one each to run the management and SQL nodes, and two computers to serve as data nodes. The purpose of the two data nodes is to provide redundancy; the management node must run on a separate machine to guarantee continued arbitration services in the event that one of the data nodes fails.

To provide increased throughput and high availability, you should use multiple SQL nodes (MySQL Servers connected to the cluster). It is also possible (although not strictly necessary) to run multiple management servers.

30.10.5: What do the different computers do in a MySQL Cluster?

A MySQL Cluster has both a physical and logical organization, with computers being the physical elements. The logical or functional elements of a cluster are referred to as *nodes*, and a computer housing a cluster node is sometimes referred to as a *cluster host*. There are three types of nodes, each corresponding to a specific role within the cluster. These are:

- **Management node (MGM node):** Provides management services for the cluster as a whole, including startup, shutdown, backups, and configuration data for the other nodes. The management node server is implemented as the application `ndb_mgmd`; the management client used to control MySQL Cluster via the MGM node is `ndb_mgm`.
- **Data node:** Stores and replicates data. Data node functionality is handled by an instance of the NDB data node process `ndbd`.
- **SQL node:** This is simply an instance of MySQL Server (`mysqld`) that is built with support for the **NDB Cluster** storage engine and started with the `--ndb-cluster` option to enable the engine.

30.10.6: With which operating systems can I use Cluster?

MySQL Cluster is supported on most Unix-like operating systems, including Linux, Mac OS X, Solaris, BSD, HP-UX, AIX, and IRIX, among others, as well as Novell Netware. Cluster is not supported for Windows at this time. However, we are working to add Cluster support for other platforms, including Windows, and our goal is to offer MySQL Cluster on all platforms for which MySQL itself is supported.

For more detailed information concerning the level of support which is offered for MySQL Cluster on various operating system versions, OS distributions, and hardware platforms, please refer to <http://www.mysql.com/support/supportedplatforms/cluster.html>.

30.10.7: What are the hardware requirements for running MySQL Cluster?

Cluster should run on any platform for which NDB-enabled binaries are available. Naturally, faster CPUs and more memory will improve performance, and 64-bit CPUs will likely be more effective than 32-bit processors. There must be sufficient memory on machines used for data nodes to hold each node's share of the database (see *How much RAM do I Need?* for more information). Nodes can communicate via a standard TCP/IP network and hardware. For SCI support, special networking hardware is required.

30.10.8: How much RAM do I need? Is it possible to use disk memory at all?

Previous to MySQL 5.1, Cluster was in-memory only. This meant that all table data (including indexes) was stored in RAM. If your data took up 1GB of space and you wanted to replicate it once in the cluster, you needed 2GB of memory to do so (1 GB per replica). This was in addition to the memory required by the operating system and any applications running on the cluster computers. This is still true of in-memory tables.

If a data node's memory usage exceeds what is available in RAM, then the system will attempt to use swap space up to the limit set for **DataMemory**. However, this will at best result in severely degraded performance, and may cause the node to be dropped due to slow response time (missed heartbeats). We do not recommend on relying on disk swapping in a production environment for this reason. In

any case, once the [DataMemory](#) limit is reached, any operations requiring additional memory (such as inserts) will fail.

[NDB Cluster](#) in MySQL 5.1 includes support for Disk Data, which helps to alleviate these issues. See [Section 16.11, “MySQL Cluster Disk Data Tables”](#), for more information.

You can use the following formula for obtaining a rough estimate of how much RAM is needed for each data node in the cluster:

```
(SizeofDatabase × NumberOfReplicas × 1.1 ) / NumberOfDataNodes
```

To calculate the memory requirements more exactly requires determining, for each table in the cluster database, the storage space required per row (see [Section 10.5, “Data Type Storage Requirements”](#), for details), and multiplying this by the number of rows. You must also remember to account for any column indexes as follows:

- Each primary key or hash index created for an [NDBCluster](#) table requires 21–25 bytes per record. These indexes use [IndexMemory](#).
- Each ordered index requires 10 bytes storage per record, using [DataMemory](#).
- Creating a primary key or unique index also creates an ordered index, unless this index is created with [USING HASH](#). In other words:
 - A primary key or unique index on a Cluster table normally takes up 31 to 35 bytes per record.
 - However, if the primary key or unique index is created with [USING HASH](#), then it requires only 21 to 25 bytes per record.

Note that creating MySQL Cluster tables with [USING HASH](#) for all primary keys and unique indexes will generally cause table updates to run more quickly — in some cases by as much as 20 to 30 percent faster than updates on tables where [USING HASH](#) was not used in creating primary and unique keys. This is due to the fact that less memory is required (because no ordered indexes are created), and that less CPU must be utilized (because fewer indexes must be read and possibly updated). However, it also means that queries that could otherwise use range scans must be satisfied by other means, which can result in slower selects.

When calculating Cluster memory requirements, you may find useful the [ndb_size.pl](#) utility which is available in recent MySQL 5.1 releases. This Perl script connects to a current (non-Cluster) MySQL database and creates a report on how much space that database would require if it used the [NDBCluster](#) storage engine. For more information, see [Section 16.9.15, “ndb_size.pl — NDBCluster Size Requirement Estimator”](#).

It is especially important to keep in mind that *every MySQL Cluster table must have a primary key*. The [NDB](#) storage engine creates a primary key automatically if none is defined, and this primary key is created without [USING HASH](#).

There is no easy way to determine exactly how much memory is being used for storage of Cluster indexes at any given time; however, warnings are written to the Cluster log when 80% of available [DataMemory](#) or [IndexMemory](#) is in use, and again when use reaches 85%, 90%, and so on.

30.10.9: What filesystems can I use with MySQL Cluster? What about network filesystems or network shares?

Generally, any filesystem that is native to the host operating system should work well with MySQL Cluster. If you find that a given filesystem works particularly well (or not so especially well) with MySQL Cluster, we invite you to discuss your findings in the [MySQL Cluster Forums](#).

We do not test MySQL Cluster with [FAT](#) or [VFAT](#) filesystems on Linux. Because of this, and due to the fact that these are not very useful for any purpose other than sharing disk partitions between Linux and Windows operating systems on multi-boot computers, we do not recommend their use with MySQL Cluster.

MySQL Cluster is implemented as a shared-nothing solution; the idea behind this is that the failure of a single piece of hardware should not cause the failure of multiple cluster nodes, or possibly even the failure of the cluster as a whole. For this reason, the use of network shares or network filesystems is not supported for MySQL Cluster. This also applies to shared storage devices such as SANs.

30.10.10: Can I run MySQL Cluster nodes inside virtual machines (such as those created by VMWare, Parallels, or Xen)?

This is possible but not recommended for a production environment.

We have found that running MySQL Cluster processes inside a virtual machine can give rise to issues with timing and disk subsystems that have a strong negative impact on the operation of the cluster. The behavior of the cluster is often unpredictable in these cases.

If the issue can be reproduced outside the virtual environment, then we may be able to provide assistance. Otherwise, we cannot support it at this time.

30.10.11: I'm trying to populate a Cluster database. The loading process terminates prematurely and I get an error message like this one:

```
ERROR 1114: THE TABLE 'MY_CLUSTER_TABLE' IS FULL
```

Why is this happening?

The cause is very likely to be that your setup does not provide sufficient RAM for all table data and all indexes, *including the primary key required by the [NDB](#) storage engine and automatically created in the event that the table definition does not include the definition of a primary key.*

It is also worth noting that all data nodes should have the same amount of RAM, since no data node in a cluster can use more memory than the least amount available to any individual data node. In other words, if there are four computers hosting Cluster data nodes, and three of these have 3GB of RAM available to store Cluster data while the remaining data node has only 1GB RAM, then each data node can devote only 1GB to clustering.

30.10.12: MySQL Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?

It is very unlikely that a cluster would perform reliably under such conditions, as MySQL Cluster was designed and implemented with the assumption that it would be run under conditions guaranteeing dedicated high-speed connectivity such as that found in a LAN setting using 100 Mbps or gigabit Ethernet — preferably the latter. We neither test nor warrant its performance using anything slower than this.

Also, it is extremely important to keep in mind that communications between the nodes in a MySQL Cluster are not secure; they are neither encrypted nor safeguarded by any other protective mechanism. The most secure configuration for a cluster is in a private network behind a firewall, with no direct access to any Cluster data or management nodes from outside. (For SQL nodes, you should take the same precautions as you would with any other instance of the MySQL server.)

30.10.13: Do I have to learn a new programming or query language to use MySQL Cluster?

No. Although some specialized commands are used to manage and configure the cluster itself, only standard (My)SQL queries and commands are required for the following operations:

- Creating, altering, and dropping tables (including Disk Data tables and related objects)
- Inserting, updating, and deleting table data
- Creating, changing, and dropping primary and unique indexes

Some specialized configuration parameters and files are required to set up a MySQL Cluster — see [Section 16.4.4, “Configuration File”](#), for information about these.

A few simple commands are used in the MySQL Cluster management client for tasks such as starting and stopping cluster nodes. See [Section 16.7.2, “Commands in the MySQL Cluster Management Client”](#).

30.10.14: How do I find out what an error or warning message means when using MySQL Cluster?

There are two ways in which this can be done:

- From within the `mysql` client, use `SHOW ERRORS` or `SHOW WARNINGS` immediately upon being notified of the error or warning condition. Errors and warnings also be displayed in MySQL Query Browser.
- From a system shell prompt, use `pererror --ndb error_code`.

30.10.15: Is MySQL Cluster transaction-safe? What isolation levels are supported?

Yes: For tables created with the [NDB](#) storage engine, transactions are supported. In MySQL 5.1, Cluster supports only the [READ COM-](#)

MITTED transaction isolation level.

30.10.16: What storage engines are supported by MySQL Cluster?

Clustering in MySQL is supported only by the **NDB** storage engine. That is, in order for a table to be shared between nodes in a cluster, it must be created using **ENGINE=NDB** (or **ENGINE=NDBCLUSTER**, which is equivalent).

It is possible to create tables using other storage engines (such as **MyISAM** or **InnoDB**) on a MySQL server being used for clustering, but these non-**NDB** tables will **not** participate in the cluster; they are local to the individual MySQL server instance on which they are created.

30.10.17: Which versions of the MySQL software support Cluster? Do I have to compile from source?

Cluster is supported in all server binaries in the 5.1 release series for operating systems on which MySQL Cluster is available. See [Section 5.2, “mysqld — The MySQL Server”](#). You can determine whether your server has NDB support using either the **SHOW VARIABLES LIKE 'have_%'** or **SHOW ENGINES** statement.

You can also obtain NDB support by compiling MySQL from source, but it is not necessary to do so simply to use MySQL Cluster. To download the latest binary, RPM, or source distribution in the MySQL 5.1 series, visit <http://dev.mysql.com/downloads/mysql/5.1.html>.

30.10.18: In the event of a catastrophic failure — say, for instance, the whole city loses power and my UPS fails — would I lose all my data?

All committed transactions are logged. Therefore, although it is possible that some data could be lost in the event of a catastrophe, this should be quite limited. Data loss can be further reduced by minimizing the number of operations per transaction. (It is not a good idea to perform large numbers of operations per transaction in any case.)

30.10.19: Is it possible to use **FULLTEXT** indexes with Cluster?

FULLTEXT indexing is not supported by the **NDB** storage engine in MySQL 5.1, or by any storage engine other than **MyISAM**. We are working to add this capability in a future release.

30.10.20: Can I run multiple nodes on a single computer?

It is possible but not advisable. One of the chief reasons to run a cluster is to provide redundancy. To enjoy the full benefits of this redundancy, each node should reside on a separate machine. If you place multiple nodes on a single machine and that machine fails, you lose all of those nodes. Given that MySQL Cluster can be run on commodity hardware loaded with a low-cost (or even no-cost) operating system, the expense of an extra machine or two is well worth it to safeguard mission-critical data. It also worth noting that the requirements for a cluster host running a management node are minimal. This task can be accomplished with a 200 MHz Pentium CPU and sufficient RAM for the operating system plus a small amount of overhead for the **ndb_mgmd** and **ndb_mgm** processes.

It is acceptable to run multiple cluster data nodes on a single host for learning about MySQL Cluster, or for testing purposes; however, this is *not* supported for production use.

30.10.21: Can I add nodes to a cluster without restarting it?

Not at present. A simple restart is all that is required for adding new MGM or SQL nodes to a Cluster. When adding data nodes the process is more complex, and requires the following steps:

1. Make a complete backup of all Cluster data.
2. Completely shut down the cluster and all cluster node processes.
3. Restart the cluster, using the **--initial** startup option.

Warning

Never use the **--initial** when starting **ndbd** except when necessary to clear the data node filesystem. See [Section 16.6.5.2, “Command Options for ndbd”](#), for information about when this is required.

4. Restore all cluster data from the backup.

In a future MySQL Cluster release series, we hope to implement a “hot” reconfiguration capability for MySQL Cluster to minimize (if not eliminate) the requirement for restarting the cluster when adding new nodes. However, this is not planned for MySQL 5.1.

30.10.22: Are there any limitations that I should be aware of when using MySQL Cluster?

Limitations on [NDB](#) tables in MySQL 5.1 include:

- Temporary tables are not supported; a `CREATE TEMPORARY TABLE` statement using `ENGINE=NDB` or `ENGINE=NDBCLUSTER` fails with an error.
- The only types of user-defined partitioning supported for [NDB](#) tables are [KEY](#) and [LINEAR KEY](#). (Beginning with MySQL 5.1.12, attempting to create an [NDB](#) table using any other partitioning type fails with an error.)
- [FULLTEXT](#) indexes and index prefixes are not supported. Only complete columns may be indexed.
- Spatial data types are not supported. See [Chapter 18, Spatial Extensions](#).
- Only complete rollbacks for transactions are supported. Partial rollbacks and rollbacks to save points are not supported.
- The maximum number of attributes allowed per table is 128, and attribute names cannot be any longer than 31 characters. For each table, the maximum combined length of the table and database names is 122 characters.
- The maximum size for a table row is 8 kilobytes, not counting [BLOB](#) values. There is no set limit for the number of rows per table. Table size limits depend on a number of factors, in particular on the amount of RAM available to each data node.
- The [NDB](#) engine does not support foreign key constraints. As with [MyISAM](#) tables, these are ignored.

For a complete listing of limitations in MySQL Cluster, see [Section 16.13, “Known Limitations of MySQL Cluster”](#).

30.10.23: How do I import an existing MySQL database into a cluster?

You can import databases into MySQL Cluster much as you would with any other version of MySQL. Other than the limitations mentioned elsewhere in this FAQ and in [Section 16.13, “Known Limitations of MySQL Cluster”](#), the only other special requirement is that any tables to be included in the cluster must use the [NDB](#) storage engine. This means that the tables must be created with `ENGINE=NDB` or `ENGINE=NDBCLUSTER`.

It is also possible to convert existing tables using other storage engines to [NDB Cluster](#) using one or more `ALTER TABLE` statement, but this requires an additional workaround. See [Section 16.13, “Known Limitations of MySQL Cluster”](#), for details.

30.10.24: How do cluster nodes communicate with one another?

Cluster nodes can communicate via any of three different protocols: TCP/IP, SHM (shared memory), and SCI (Scalable Coherent Interface). Where available, SHM is used by default between nodes residing on the same cluster host; however, this is considered experimental in MySQL 5.1. SCI is a high-speed (1 gigabit per second and higher), high-availability protocol used in building scalable multi-processor systems; it requires special hardware and drivers. See [Section 16.12, “Using High-Speed Interconnects with MySQL Cluster”](#), for more about using SCI as a transport mechanism in MySQL Cluster.

30.10.25: What is an arbitrator?

If one or more nodes in a cluster fail, it is possible that not all cluster nodes will be able to “see” one another. In fact, it is possible that two sets of nodes might become isolated from one another in a network partitioning, also known as a “split brain” scenario. This type of situation is undesirable because each set of nodes tries to behave as though it is the entire cluster.

When cluster nodes go down, there are two possibilities. If more than 50% of the remaining nodes can communicate with each other, we have what is sometimes called a “majority rules” situation, and this set of nodes is considered to be the cluster. The arbitrator comes into play when there is an even number of nodes: in such cases, the set of nodes to which the arbitrator belongs is considered to be the cluster, and nodes not belonging to this set are shut down.

The preceding information is somewhat simplified. A more complete explanation taking into account node groups follows:

When all nodes in at least one node group are alive, network partitioning is not an issue, because no one portion of the cluster can form a functional cluster. The real problem arises when no single node group has all its nodes alive, in which case network partitioning (the “split-brain” scenario) becomes possible. Then an arbitrator is required. All cluster nodes recognize the same node as the arbitrator, which is normally the management server; however, it is possible to configure any of the MySQL Servers in the cluster to act as the arbitrator instead. The arbitrator accepts the first set of cluster nodes to contact it, and tells the remaining set to shut down. Arbitrator selection is controlled by the [ArbitrationRank](#) configuration parameter for MySQL Server and management server nodes. (See [Section 16.4.4.4, “Defining the Management Server”](#), for details.) It should also be noted that the role of arbitrator does not in and of itself

impose any heavy demands upon the host so designated, and thus the arbitrator host does not need to be particularly fast or to have extra memory especially for this purpose.

30.10.26: What data types are supported by MySQL Cluster?

MySQL Cluster supports all of the usual MySQL data types, with the exception of those associated with MySQL's spatial extensions. (See [Chapter 18, *Spatial Extensions*](#).) In addition, there are some differences with regard to indexes when used with NDB tables. **Note:** MySQL Cluster Disk Data tables (that is, tables created with `TABLESPACE ... STORAGE DISK ENGINE=NDBCLUSTER`) have only fixed-width rows. This means that (for example) each Disk Data table record containing a `VARCHAR(255)` column requires space for 255 characters (as required for the character set and collation being used for the table), regardless of the actual number of characters stored therein.

See [Section 16.13, “Known Limitations of MySQL Cluster”](#), for more information about these issues.

30.10.27: How do I start and stop MySQL Cluster?

It is necessary to start each node in the cluster separately, in the following order:

1. Start the management node with the `ndb_mgmd` command.
2. Start each data node with the `ndbd` command.
3. Start each MySQL server (SQL node) using `mysqld_safe --user=mysql &`.

Each of these commands must be run from a system shell on the machine housing the affected node. (You do not have to be physically present at the machine — a remote login shell can be used for this purpose.) You can verify that the cluster is running by starting the MGM management client `ndb_mgm` on the machine housing the MGM node and issuing the `SHOW` or `ALL STATUS` command.

To shut down a running cluster, issue the command `SHUTDOWN` in the MGM client. Alternatively, you may enter the following command in a system shell on the machine hosting the MGM node:

```
shell> ndb_mgm -e "SHUTDOWN"
```

(Note that the quotation marks are optional here; the `SHUTDOWN` command itself is not case-sensitive.)

Either of these commands causes the `ndb_mgm`, `ndb_mgm`, and any `ndbd` processes to terminate gracefully. MySQL servers running as Cluster SQL nodes can be stopped using `mysqldadmin shutdown`.

For more information, see [Section 16.7.2, “Commands in the MySQL Cluster Management Client”](#), and [Section 16.3.6, “Safe Shut-down and Restart”](#).

30.10.28: What happens to cluster data when the cluster is shut down?

The data that was held in memory by the cluster's data nodes is written to disk, and is reloaded into memory the next time that the cluster is started.

30.10.29: Is it helpful to have more than one management node for a cluster?

It can be helpful as a fail-safe. Only one MGM node controls the cluster at any given time, but it is possible to configure one MGM as primary, and one or more additional management nodes to take over in the event that the primary MGM node fails.

See [Section 16.4.4, “Configuration File”](#), for information on how to configure MySQL Cluster management nodes.

30.10.30: Can I mix different kinds of hardware and operating systems in one MySQL Cluster?

Yes, so long as all machines and operating systems have the same “endianness” (all big-endian or all little-endian). It is also possible to use different MySQL Cluster releases on different nodes. However, we recommend this be done only as part of a rolling upgrade procedure (see [Section 16.5.1, “Performing a Rolling Restart of the Cluster”](#)).

30.10.31: Can I run two data nodes on a single host? Two SQL nodes?

Yes, it is possible to do this. In the case of multiple data nodes, it is advisable (but not required) for each node to use a different data directory. If you want to run multiple SQL nodes on one machine, each instance of `mysqld` must use a different TCP/IP port. However, *running more than one cluster node of a given type per machine is not supported for production use.*

30.10.32: Can I use hostnames with MySQL Cluster?

Yes, it is possible to use DNS and DHCP for cluster hosts. However, if your application requires “five nines” availability, we recommend using fixed IP addresses. Making communication between Cluster hosts dependent on services such as DNS and DHCP introduces additional points of failure, and the fewer of these, the better.

30.10.33: How do I handle MySQL users in a Cluster having multiple MySQL servers?

MySQL user accounts and privileges are not automatically propagated between different MySQL servers accessing the same MySQL Cluster. Therefore, you must make sure that these are copied between the SQL nodes yourself.

A.11. MySQL 5.1 FAQ — MySQL Chinese, Japanese, and Korean Character Sets

This set of Frequently Asked Questions derives from the experience of MySQL's Support and Development groups in handling many inquiries about CJK (Chinese-Japanese-Korean) issues.

Questions

- [30.11.1](#): I have inserted CJK characters into my table. Why does `SELECT` display them as “?” characters?
- [30.11.2](#): What GB (Chinese) character sets does MySQL support?
- [30.11.3](#): What problems should I be aware of when working with the Big5 Chinese character set?
- [30.11.4](#): Why do Japanese character set conversions fail?
- [30.11.5](#): What should I do if I want to convert SJIS `81CA` to `cp932`?
- [30.11.6](#): How does MySQL represent the Yen (¥) sign?
- [30.11.7](#): Do MySQL plan to make a separate character set where `5C` is the Yen sign, as at least one other major DBMS does?
- [30.11.8](#): Of what issues should I be aware when working with Korean character sets in MySQL?
- [30.11.9](#): Why do I get `DATA TRUNCATED` error messages?
- [30.11.10](#): Why does my GUI front end or browser not display CJK characters correctly in my application using Access, PHP, or another API?
- [30.11.11](#): I've upgraded to MySQL 5.1. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?
- [30.11.12](#): Why do some `LIKE` and `FULLTEXT` searches with CJK characters fail?
- [30.11.13](#): What CJK character sets are available in MySQL?
- [30.11.14](#): How do I know whether character `X` is available in all character sets?
- [30.11.15](#): Why don't CJK strings sort correctly in Unicode? (I)
- [30.11.16](#): Why don't CJK strings sort correctly in Unicode? (II)
- [30.11.17](#): Why are my supplementary characters rejected by MySQL?
- [30.11.18](#): Shouldn't it be “CJKV”?
- [30.11.19](#): Does MySQL allow CJK characters to be used in database and table names?
- [30.11.20](#): Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?
- [30.11.21](#): Where can I get help with CJK and related issues in MySQL?

Questions and Answers

30.11.1: I have inserted CJK characters into my table. Why does `SELECT` display them as “?” characters?

This problem is usually due to a setting in MySQL that doesn't match the settings for the application program or the operating system. Here are some common steps for correcting these types of issues:

- *Be certain of what MySQL version you are using.*

Use the statement `SELECT VERSION();` to determine this.

- *Make sure that the database is actually using the desired character set.*

People often think that the client character set is always the same as either the server character set or the character set used for display purposes. However, both of these are false assumptions. You can make sure by checking the result of `SHOW CREATE TABLE tablename` or — better — yet by using this statement:

```
SELECT character_set_name, collation_name
FROM information_schema.columns
WHERE table_schema = your_database_name
AND table_name = your_table_name
AND column_name = your_column_name;
```

- *Determine the hexadecimal value of the character or characters that are not being displayed correctly.*

You can obtain this information for a column *column_name* in the table *table_name* using the following query:

```
SELECT HEX(column_name)
FROM table_name;
```

`3F` is the encoding for the `?` character; this means that `?` is the character actually stored in the column. This most often happens because of a problem converting a particular character from your client character set to the target character set.

- *Make sure that a round trip possible — that is, when you select *literal* (or *_introducer hexadecimal-value*), you obtain *literal* as a result.*

For example, the Japanese *Katakana* character *Pe* (`#`) exists in all CJK character sets, and has the code point value (hexadecimal coding) `0x30da`. To test a round trip for this character, use this query:

```
SELECT '#' AS '#';          /* or SELECT _ucs2 0x30da; */
```

If the result is not also `#`, then the round trip has failed.

For bug reports regarding such failures, we might ask you to follow up with `SELECT HEX('#');`. Then we can determine whether the client encoding is correct.

- *Make sure that the problem is not with the browser or other application, rather than with MySQL.*

Use the `mysql` client program (on Windows: `mysql.exe`) to accomplish this task. If `mysql` displays correctly but your application doesn't, then your problem is probably due to system settings.

To find out what your settings are, use the `SHOW VARIABLES` statement, whose output should resemble what is shown here:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | utf8 |
| character_set_connection | utf8 |
| character_set_database | latin1 |
| character_set_filesystem | binary |
| character_set_results | utf8 |
| character_set_server | latin1 |
| character_set_system | utf8 |
| character_sets_dir | /usr/local/mysql/share/mysql/charsets/ |
+-----+-----+
8 rows in set (0.03 sec)
```

These are typical character-set settings for an international-oriented client (notice the use of `utf8` Unicode) connected to a server in the West (`latin1` is a West Europe character set and a default for MySQL).

Although Unicode (usually the `utf8` variant on Unix, and the `ucs2` variant on Windows) is preferable to Latin, it's often not what

your operating system utilities support best. Many Windows users find that a Microsoft character set, such as [cp932](#) for Japanese Windows, is what's suitable.

If you cannot control the server settings, and you have no idea what your underlying computer is, then try changing to a common character set for the country that you're in ([euckr](#) = Korea; [gb2312](#) or [gbk](#) = People's Republic of China; [big5](#) = Taiwan; [sjis](#), [ujis](#), [cp932](#), or [eucjpm](#)s = Japan; [ucs2](#) or [utf8](#) = anywhere). Usually it is necessary to change only the client and connection and results settings. There is a simple statement which changes all three at once: [SET NAMES](#). For example:

```
SET NAMES 'big5';
```

Once the setting is correct, you can make it permanent by editing [my.cnf](#) or [my.ini](#). For example you might add lines looking like these:

```
[mysqld]
character-set-server=big5
[client]
default-character-set=big5
```

It is also possible that there are issues with the API configuration setting being used in your application; see *Why does my GUI front end or browser not display CJK characters correctly...*? for more information.

30.11.2: What GB (Chinese) character sets does MySQL support?

MySQL supports the two common variants of the *GB* (*Guojia Biaozhun*, or *National Standard*) character sets which are official in the People's Republic of China: [gb2312](#) and [gbk](#). Sometimes people try to insert [gbk](#) characters into [gb2312](#), and it works most of the time because [gbk](#) is a superset of [gb2312](#) — but eventually they try to insert a rarer Chinese character and it doesn't work. (See [Bug#16072](#) for an example).

Here, we try to clarify exactly what characters are legitimate in [gb2312](#) or [gbk](#), with reference to the official documents. Please check these references before reporting [gb2312](#) or [gbk](#) bugs.

- For a complete listing of the [gb2312](#) characters, ordered according to the [gb2312_chinese_ci](#) collation: http://d.udm.net/bar/~bar/charts/gb2312_chinese_ci.html.
- MySQL's [gbk](#) is in reality “Microsoft code page 936”. This differs from the official [gbk](#) for characters [A1A4](#) (middle dot), [A1AA](#) (em dash), [A6E0-A6F5](#), and [A8BB-A8C0](#). For a listing of the differences, see <http://recode.progiels-bpi.ca/showfile.html?name=dist/libiconv/gbk.h>.
- For a listing of [gbk](#)/Unicode mappings, see <http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP936.TXT>.
- For MySQL's listing of [gbk](#) characters, see http://d.udm.net/bar/~bar/charts/gbk_chinese_ci.html.

30.11.3: What problems should I be aware of when working with the Big5 Chinese character set?

MySQL supports the Big5 character set which is common in Hong Kong and Taiwan (Republic of China). MySQL's [big5](#) is in reality Microsoft code page 950, which is very similar to the original [big5](#) character set. We changed to this character set starting with MySQL version 4.1.16 / 5.0.16 (as a result of [Bug#12476](#)). For example, the following statements work in current versions of MySQL, but not in old versions:

```
mysql> CREATE TABLE big5 (BIG5 CHAR(1) CHARACTER SET BIG5);
Query OK, 0 rows affected (0.13 sec)

mysql> INSERT INTO big5 VALUES (0xf9dc);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM big5;
+-----+
| big5 |
+-----+
| #    |
+-----+
1 row in set (0.02 sec)
```

A feature request for adding [HKSCS](#) extensions has been filed. People who need this extension may find the suggested patch for [Bug#13577](#) to be of interest.

30.11.4: Why do Japanese character set conversions fail?

MySQL supports the `sjis`, `ujis`, `cp932`, and `eucjpm`s character sets, as well as Unicode. A common need is to convert between character sets. For example, there might be a Unix server (typically with `sjis` or `ujis`) and a Windows client (typically with `cp932`).

In the following conversion table, the `ucs2` column represents the source, and the `sjis`, `cp932`, `ujis`, and `eucjpm`s columns represent the destinations — that is, the last 4 columns provide the hexadecimal result when we use `CONVERT(ucs2)` or we assign a `ucs2` column containing the value to an `sjis`, `cp932`, `ujis`, or `eucjpm`s column.

Character Name	ucs2	sjis	cp932	ujis	eucjpm
BROKEN BAR	00A6	3F	3F	8FA2C3	3F
FULLWIDTH BROKEN BAR	FFE4	3F	FA55	3F	8FA2
YEN SIGN	00A5	3F	3F	20	3F
FULLWIDTH YEN SIGN	FFE5	818F	818F	A1EF	3F
TILDE	007E	7E	7E	7E	7E
OVERLINE	203E	3F	3F	20	3F
HORIZONTAL BAR	2015	815C	815C	A1BD	A1BD
EM DASH	2014	3F	3F	3F	3F
REVERSE SOLIDUS	005C	815F	5C	5C	5C
FULLWIDTH ""	FF3C	3F	815F	3F	A1C0
WAVE DASH	301C	8160	3F	A1C1	3F
FULLWIDTH TILDE	FF5E	3F	8160	3F	A1C1
DOUBLE VERTICAL LINE	2016	8161	3F	A1C2	3F
PARALLEL TO	2225	3F	8161	3F	A1C2
MINUS SIGN	2212	817C	3F	A1DD	3F
FULLWIDTH HYPHEN-MINUS	FF0D	3F	817C	3F	A1DD
CENT SIGN	00A2	8191	3F	A1F1	3F
FULLWIDTH CENT SIGN	FFE0	3F	8191	3F	A1F1
POUND SIGN	00A3	8192	3F	A1F2	3F
FULLWIDTH POUND SIGN	FFE1	3F	8192	3F	A1F2
NOT SIGN	00AC	81CA	3F	A2CC	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA	3F	A2CC

Now consider this portion of the table:

	ucs2	sjis	cp932
NOT SIGN	00AC	81CA	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA

This means that MySQL converts the `NOT SIGN` (Unicode `U+00AC`) to `sjis` code point `0x81CA` and to `cp932` code point `3F`. (`3F` is the question mark (“?”) — this is what is always used when the conversion cannot be performed.

30.11.5: What should I do if I want to convert SJIS 81CA to cp932?

Our answer is: “?”. There are serious complaints about this: many people would prefer a “loose” conversion, so that `81CA` (`NOT SIGN`) in `sjis` becomes `81CA` (`FULLWIDTH NOT SIGN`) in `cp932`. We are considering a change to this behavior.

30.11.6: How does MySQL represent the Yen (¥) sign?

A problem arises because some versions of Japanese character sets (both `sjis` and `euc`) treat `5C` as a *reverse solidus* (`\` — also known as a backslash), and others treat it as a yen sign (¥).

MySQL follows only one version of the JIS (Japanese Industrial Standards) standard description. In MySQL, `5C` is always the *reverse solidus* (`\`).

30.11.7: Do MySQL plan to make a separate character set where 5C is the Yen sign, as at least one other major DBMS does?

This is one possible solution to the Yen sign issue; however, this will not happen in MySQL 5.1 or 5.2.

30.11.8: Of what issues should I be aware when working with Korean character sets in MySQL?

In theory, while there have been several versions of the `euckr` (*Extended Unix Code Korea*) character set, only one problem has been noted.

We use the “ASCII” variant of EUC-KR, in which the code point `0x5c` is REVERSE SOLIDUS, that is `\`, instead of the “KS-Roman” variant of EUC-KR, in which the code point `0x5c` is `WON SIGN(#)`. This means that you cannot convert Unicode `U+20A9` to `euckr`:

```
mysql> SELECT
-> CONVERT('#' USING euckr) AS euckr,
-> HEX(CONVERT('#' USING euckr)) AS hexeuckr;
+-----+-----+
| euckr | hexeuckr |
+-----+-----+
| ?    | 3F      |
+-----+-----+
1 row in set (0.00 sec)
```

MySQL's graphic Korean chart is here: http://d.udm.net/bar/~bar/charts/euckr_korean_ci.html.

30.11.9: Why do I get DATA TRUNCATED error messages?

For illustration, we'll create a table with one Unicode (`ucs2`) column and one Chinese (`gb2312`) column.

```
mysql> CREATE TABLE ch
-> (ucs2 CHAR(3) CHARACTER SET ucs2,
-> gb2312 CHAR(3) CHARACTER SET gb2312);
Query OK, 0 rows affected (0.05 sec)
```

We'll try to place the rare character `#` in both columns.

```
mysql> INSERT INTO ch VALUES ('A#B', 'A#B');
Query OK, 1 row affected, 1 warning (0.00 sec)
```

Ah, there's a warning. Let's see what it is.

```
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1265 | Data truncated for column 'gb2312' at row 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

So it's a warning about the `gb2312` column only.

```
mysql> SELECT ucs2, HEX(ucs2), gb2312, HEX(gb2312) FROM ch;
+-----+-----+-----+-----+
| ucs2 | HEX(ucs2) | gb2312 | HEX(gb2312) |
+-----+-----+-----+-----+
| A#B | 00416C4C0042 | A?B | 413F42 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

There are several things that need explanation here.

1. The fact that it's a “warning” rather than an “error” is characteristic of MySQL. We like to try to do what we can, to get the best fit, rather than give up.
2. The `#` character isn't in the `gb2312` character set. We described that problem earlier.
3. Admittedly the message is misleading. We didn't “truncate” in this case, we replaced with a question mark. We've had a complaint about this message (See [Bug#9337](#)). But until we come up with something better, just accept that error/warning code 2165 can mean a variety of things.
4. With `SQL_MODE=TRADITIONAL`, there would be an error message, but instead of error 2165 you would see: `ERROR 1406 (22001): Data too long for column 'gb2312' at row 1`.

30.11.10: Why does my GUI front end or browser not display CJK characters correctly in my application using Access, PHP, or

another API?

Obtain a direct connection to the server using the `mysql` client (Windows: `mysql.exe`), and try the same query there. If `mysql` responds correctly, then the trouble may be that your application interface requires initialization. Use `mysql` to tell you what character set or sets it uses with the statement `SHOW VARIABLES LIKE 'char%';`. If you are using Access, then you are most likely connecting with MyODBC. In this case, you should check [Section 26.1.3, “Connector/ODBC Configuration”](#). If, for instance, you use `big5`, you would enter `SET NAMES 'big5'`. (Note that no `;` is required in this case). If you are using ASP, you might need to add `SET NAMES` in the code. Here is an example that has worked in the past:

```
<%
Session.CodePage=0
Dim strConnection
Dim Conn
strConnection="driver={MySQL ODBC 3.51 Driver};server=server;uid=username;" \
& "pwd=password;database=database;stmt=SET NAMES 'big5';"
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open strConnection
%>
```

In much the same way, if you are using any character set other than `latin1` with Connector/NET, then you must specify the character set in the connection string. See [Section 26.2.5.1, “Connecting to MySQL Using Connector/NET”](#), for more information.

If you are using PHP, try this:

```
<?php
$link = mysql_connect($host, $usr, $pwd);
mysql_select_db($db);

if( mysql_error() ) { print "Database ERROR: " . mysql_error(); }
mysql_query("SET NAMES 'utf8'", $link);
?>
```

In this case, we used `SET NAMES` to change `character_set_client` and `character_set_connection` and `character_set_results`.

We encourage the use of the newer `mysqli` extension, rather than `mysql`. Using `mysqli`, the previous example could be rewritten as shown here:

```
<?php
$link = new mysqli($host, $usr, $pwd, $db);

if( mysqli_connect_errno() )
{
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$link->query("SET NAMES 'utf8'");
?>
```

Another issue often encountered in PHP applications has to do with assumptions made by the browser. Sometimes adding or changing a `<meta>` tag suffices to correct the problem: for example, to insure that the user agent interprets page content as `UTF-8`, you should include `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">` in the `<head>` of the HTML page.

If you are using Connector/J, see [Section 26.4.4.4, “Using Character Sets and Unicode”](#).

30.11.11: I’ve upgraded to MySQL 5.1. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?

In MySQL Version 4.0, there was a single “global” character set for both server and client, and the decision as to which character to use was made by the server administrator. This changed starting with MySQL Version 4.1. What happens now is a “handshake”, as described in [Section 9.4, “Connection Character Sets and Collations”](#):

When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and `character_set_connection` system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

The effect of this is that you cannot control the client character set by starting `mysqld` with `--character-set-server=utf8`. However, some of our Asian customers have said that prefer the MySQL 4.0 behavior. To make it possible to retain this behavior, we added a `mysqld` switch, `--character-set-client-handshake`, which can be turned off with `--skip-character-set-client-handshake`. If you start `mysqld` with `--skip-character-set-client-handshake`, then, when a client connects, it sends to the server the name of the character set

that it wants to use — however, *the server ignores this request from the client*.

By way of example, suppose that your favorite server character set is `latin1` (unlikely in a CJK area, but this is the default value). Suppose further that the client uses `utf8` because this is what the client's operating system supports. Now, start the server with `latin1` as its default character set:

```
mysqld --character-set-server=latin1
```

And then start the client with the default character set `utf8`:

```
mysql --default-character-set=utf8
```

The current settings can be seen by viewing the output of `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | utf8 |
| character_set_connection | utf8 |
| character_set_database | latin1 |
| character_set_filesystem | binary |
| character_set_results | utf8 |
| character_set_server | latin1 |
| character_set_system | utf8 |
| character_sets_dir | /usr/local/mysql/share/mysqlCharsets/ |
+-----+-----+
8 rows in set (0.01 sec)
```

Now stop the client, and then stop the server using `mysqladmin`. Then start the server again, but this time tell it to skip the handshake like so:

```
mysqld --character-set-server=utf8 --skip-character-set-client-handshake
```

Start the client with `utf8` once again as the default character set, then display the current settings:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | latin1 |
| character_set_connection | latin1 |
| character_set_database | latin1 |
| character_set_filesystem | binary |
| character_set_results | latin1 |
| character_set_server | latin1 |
| character_set_system | utf8 |
| character_sets_dir | /usr/local/mysql/share/mysqlCharsets/ |
+-----+-----+
8 rows in set (0.01 sec)
```

As you can see by comparing the differing results from `SHOW VARIABLES`, the server ignores the client's initial settings if the `--skip-character-set-client-handshake` is used.

30.11.12: Why do some `LIKE` and `FULLTEXT` searches with CJK characters fail?

There is a very simple problem with `LIKE` searches on `BINARY` and `BLOB` columns: we need to know the end of a character. With multi-byte character sets, different characters might have different octet lengths. For example, in `utf8`, `A` requires one byte but `#` requires three bytes, as shown here:

```
+-----+-----+
| OCTET_LENGTH(_utf8 'A') | OCTET_LENGTH(_utf8 '#') |
+-----+-----+
| 1 | 3 |
+-----+-----+
1 row in set (0.00 sec)
```

If we don't know where the first character ends, then we don't know where the second character begins, in which case even very simple searches such as `LIKE '_A%'` fail. The solution is to use a regular CJK character set in the first place, or to convert to a CJK character set before comparing.

This is one reason why MySQL cannot allow encodings of nonexistent characters. If it is not strict about rejecting bad input, then it has no way of knowing where characters end.

For `FULLTEXT` searches, we need to know where words begin and end. With Western languages, this is rarely a problem because most (if not all) of these use an easy-to-identify word boundary — the space character. However, this is not usually the case with Asian writing. We could use arbitrary halfway measures, like assuming that all Han characters represent words, or (for Japanese) depending on changes from Katakana to Hiragana due to grammatical endings. However, the only sure solution requires a comprehensive word list,

which means that we would have to include a dictionary in the server for each Asian language supported. This is simply not feasible.

30.11.13: What CJK character sets are available in MySQL?

The list of CJK character sets may vary depending on your MySQL version. For example, the [eucjpms](#) character set was not supported prior to MySQL 5.0.3. However, since the name of the applicable language appears in the [DESCRIPTION](#) column for every entry in the [INFORMATION_SCHEMA.CHARACTER_SETS](#) table, you can obtain a current list of all the non-Unicode CJK character sets using this query:

```
mysql> SELECT CHARACTER_SET_NAME, DESCRIPTION
-> FROM INFORMATION_SCHEMA.CHARACTER_SETS
-> WHERE DESCRIPTION LIKE '%Chinese%'
-> OR DESCRIPTION LIKE '%Japanese%'
-> OR DESCRIPTION LIKE '%Korean%'
-> ORDER BY CHARACTER_SET_NAME;
```

CHARACTER_SET_NAME	DESCRIPTION
big5	Big5 Traditional Chinese
cp932	SJIS for Windows Japanese
eucjpms	UJIS for Windows Japanese
euckr	EUC-KR Korean
gb2312	GB2312 Simplified Chinese
gbk	GBK Simplified Chinese
sjis	Shift-JIS Japanese
ujis	EUC-JP Japanese

8 rows in set (0.01 sec)

(See [Section 23.9](#), “[The INFORMATION_SCHEMA.CHARACTER_SETS Table](#)”, for more information.)

30.11.14: How do I know whether character *X* is available in all character sets?

The majority of simplified Chinese and basic non-halfwidth Japanese *Kana* characters appear in all CJK character sets. This stored procedure accepts a [UCS-2](#) Unicode character, converts it to all other character sets, and displays the results in hexadecimal.

```
DELIMITER //

CREATE PROCEDURE p_convert(ucs2_char CHAR(1) CHARACTER SET ucs2)
BEGIN
  CREATE TABLE tj
  (ucs2 CHAR(1) character set ucs2,
   utf8 CHAR(1) character set utf8,
   big5 CHAR(1) character set big5,
   cp932 CHAR(1) character set cp932,
   eucjpms CHAR(1) character set eucjpms,
   euckr CHAR(1) character set euckr,
   gb2312 CHAR(1) character set gb2312,
   gbk CHAR(1) character set gbk,
   sjis CHAR(1) character set sjis,
   ujis CHAR(1) character set ujis);

  INSERT INTO tj (ucs2) VALUES (ucs2_char);

  UPDATE tj SET utf8=ucs2,
               big5=ucs2,
               cp932=ucs2,
               eucjpms=ucs2,
               euckr=ucs2,
               gb2312=ucs2,
               gbk=ucs2,
               sjis=ucs2,
               ujis=ucs2;

  /* If there's a conversion problem, UPDATE will produce a warning. */

  SELECT hex(ucs2) AS ucs2,
         hex(utf8) AS utf8,
         hex(big5) AS big5,
         hex(cp932) AS cp932,
         hex(eucjpms) AS eucjpms,
         hex(euckr) AS euckr,
         hex(gb2312) AS gb2312,
         hex(gbk) AS gbk,
         hex(sjis) AS sjis,
         hex(ujis) AS ujis
  FROM tj;

  DROP TABLE tj;

END//
```

The input can be any single [ucs2](#) character, or it can be the code point value (hexadecimal representation) of that character. For ex-

ample, from Unicode's list of `ucs2` encodings and names (<http://www.unicode.org/Public/UNIDATA/UnicodeData.txt>), we know that the *Katakana* character *Pe* appears in all CJK character sets, and that its code point value is `0x30da`. If we use this value as the argument to `p_convert()`, the result is as shown here:

```
mysql> CALL p_convert(0x30da)//
+-----+-----+-----+-----+-----+-----+-----+-----+
| ucs2 | utf8 | big5 | cp932 | eucjpms | euckr | gb2312 | gbk | sjis | ujis |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 30DA | E3839A | C772 | 8379 | A5DA | ABDA | A5DA | A5DA | 8379 | A5DA |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.04 sec)
```

Since none of the column values is `3F` — that is, the question mark character (?) — we know that every conversion worked.

30.11.15: Why don't CJK strings sort correctly in Unicode? (I)

Sometimes people observe that the result of a `utf8_unicode_ci` or `ucs2_unicode_ci` search, or of an `ORDER BY` sort is not what they think a native would expect. Although we never rule out the possibility that there is a bug, we have found in the past that many people do not read correctly the standard table of weights for the Unicode Collation Algorithm. MySQL uses the table found at <http://www.unicode.org/Public/UCD/4.0.0/allkeys-4.0.0.txt>. This is not the first table you will find by navigating from the [uniconcode.org](http://www.unicode.org) home page, because MySQL uses the older 4.0.0 “allkeys” table, rather than the more recent 4.1.0 table. This is because we are very wary about changing ordering which affects indexes, lest we bring about situations such as that reported in [Bug#16526](#), illustrated as follows:

```
mysql> CREATE TABLE tj (s1 CHAR(1) CHARACTER SET utf8 COLLATE utf8_unicode_ci);
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO tj VALUES ('#'),('#');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM tj WHERE s1 = '#';
+-----+
| s1 |
+-----+
| # |
| # |
+-----+
2 rows in set (0.00 sec)
```

The character in the first result row is not the one that we searched for. Why did MySQL retrieve it? First we look for the Unicode code point value, which is possible by reading the hexadecimal number for the `ucs2` version of the characters:

```
mysql> SELECT s1, HEX(CONVERT(s1 USING ucs2)) FROM tj;
+-----+-----+
| s1 | HEX(CONVERT(s1 USING ucs2)) |
+-----+-----+
| # | 304C |
| # | 304B |
+-----+-----+
2 rows in set (0.03 sec)
```

Now we search for `304B` and `304C` in the `4.0.0 allkeys` table, and find these lines:

```
304B ; [.1E57.0020.000E.304B] # HIRAGANA LETTER KA
304C ; [.1E57.0020.000E.304B][.0000.0140.0002.3099] # HIRAGANA LETTER GA; QQCM
```

The official Unicode names (following the “#” mark) tell us the Japanese syllabary (Hiragana), the informal classification (letter, digit, or punctuation mark), and the Western identifier (*KA* or *GA*, which happen to be voiced and unvoiced components of the same letter pair). More importantly, the *primary weight* (the first hexadecimal number inside the square brackets) is `1E57` on both lines. For comparisons in both searching and sorting, MySQL pays attention to the primary weight only, ignoring all the other numbers. This means that we are sorting `#` and `#` correctly according to the Unicode specification. If we wanted to distinguish them, we'd have to use a non-UCA (Unicode Collation Algorithm) collation (`utf8_unicode_bin` or `utf8_general_ci`), or to compare the `HEX()` values, or use `ORDER BY CONVERT(s1 USING sjis)`. Being correct “according to Unicode” isn't enough, of course: the person who submitted the bug was equally correct. We plan to add another collation for Japanese according to the JIS X 4061 standard, in which voiced/unvoiced letter pairs like *KA/GA* are distinguishable for ordering purposes.

30.11.16: Why don't CJK strings sort correctly in Unicode? (II)

If you are using Unicode (`ucs2` or `utf8`), and you know what the Unicode sort order is (see [Section A.11, “MySQL 5.1 FAQ — MySQL Chinese, Japanese, and Korean Character Sets”](#)), but MySQL still seems to sort your table incorrectly, then you should first verify the table character set:

```
mysql> SHOW CREATE TABLE t\G
***** 1. row *****
Table: t
Create Table: CREATE TABLE `t` (
```



```
`s1` char(1) CHARACTER SET ucs2 DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Since the character set appears to be correct, let's see what information the `INFORMATION_SCHEMA.COLUMNS` table can provide about this column:

```
mysql> SELECT COLUMN_NAME, CHARACTER_SET_NAME, COLLATION_NAME
-> FROM INFORMATION_SCHEMA.COLUMNS
-> WHERE COLUMN_NAME = 's1'
-> AND TABLE_NAME = 't';
+-----+-----+-----+
| COLUMN_NAME | CHARACTER_SET_NAME | COLLATION_NAME |
+-----+-----+-----+
| s1          | ucs2               | ucs2_general_ci |
+-----+-----+-----+
1 row in set (0.01 sec)
```

(See [Section 23.3, “The INFORMATION_SCHEMA COLUMNS Table”](#), for more information.)

You can see that the collation is `ucs2_general_ci` instead of `ucs2_unicode_ci`. The reason why this is so can be found using `SHOW CHARSET`, as shown here:

```
mysql> SHOW CHARSET LIKE 'ucs2%';
+-----+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+-----+
| ucs2    | UCS-2 Unicode | ucs2_general_ci   | 2      |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

For `ucs2` and `utf8`, the default collation is “general”. To specify a Unicode collation, use `COLLATE ucs2_unicode_ci`.

30.11.17: Why are my supplementary characters rejected by MySQL?

MySQL does not support supplementary characters — that is, characters which need more than 3 bytes — for `UTF-8`. We support only what Unicode calls the *Basic Multilingual Plane / Plane 0*. Only a few very rare Han characters are supplementary; support for them is uncommon. This has led to reports such as that found in [Bug#12600](#), which we rejected as “not a bug”. With `utf8`, we must truncate an input string when we encounter bytes that we don't understand. Otherwise, we wouldn't know how long the bad multi-byte character is.

One possible workaround is to use `ucs2` instead of `utf8`, in which case the “bad” characters are changed to question marks; however, no truncation takes place. You can also change the data type to `BLOB` or `BINARY`, which perform no validity checking.

We intend at some point in the future to add support for `UTF-16`, which would solve such issues by allowing 4-byte characters. However, we have as yet set no definite timetable for doing so.

30.11.18: Shouldn't it be “CJKV”?

No. The term “CJKV” (*Chinese Japanese Korean Vietnamese*) refers to Vietnamese character sets which contain Han (originally Chinese) characters. MySQL has no plan to support the old Vietnamese script using Han characters. MySQL does of course support the modern Vietnamese script with Western characters.

[Bug#4745](#) is a request for a specialized Vietnamese collation, which we might add in the future if there is sufficient demand for it.

30.11.19: Does MySQL allow CJK characters to be used in database and table names?

This issue is fixed in MySQL 5.1, by automatically rewriting the names of the corresponding directories and files.

For example, if you create a database named `#` on a server whose operating system does not support CJK in directory names, MySQL creates a directory named `@0w@00a5@00ae.` which is just a fancy way of encoding `E6A5AE` — that is, the Unicode hexadecimal representation for the `#` character. However, if you run a `SHOW DATABASES` statement, you can see that the database is listed as `#`.

30.11.20: Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?

A Simplified Chinese version of the Manual, current for MySQL 5.1.12, can be found at <http://dev.mysql.com/doc/#chinese-5.1>. The Japanese translation of the MySQL 4.1 manual can be downloaded from <http://dev.mysql.com/doc/#japanese-4.1>.

30.11.21: Where can I get help with CJK and related issues in MySQL?

The following resources are available:

- A listing of MySQL user groups can be found at <http://dev.mysql.com/user-groups/>.
- You can contact a sales engineer at the MySQL KK Japan office using any of the following:

Tel: +81(0)3-5326-3133
Fax: +81(0)3-5326-3001
Email: dsaito@mysql.com

- View feature requests relating to character set issues at <http://tinyurl.com/y6xcuf>.
- Visit the MySQL [Character Sets](#), [Collation](#), [Unicode Forum](#). We are also in the process of adding foreign-language forums at <http://forums.mysql.com/>.

A.12. MySQL 5.1 FAQ — Connectors & APIs

For common questions, issues, and answers relating to the MySQL Connectors and other APIs, see the following areas of the Manual:

- [Section 25.2.14, “Common Questions and Problems When Using the C API”](#)
- [Section 25.3.1, “Common Problems with MySQL and PHP”](#)
- [Section 26.1.6, “Connector/ODBC Notes and Tips”](#)
- [Section 26.2.5, “Connector/NET Notes and Tips”](#)
- [Section 26.4.5, “Connector/J Notes and Tips”](#)
- [Section 26.5.5, “Connector/MXJ Notes and Tips”](#)

A.13. MySQL 5.1 FAQ — Replication

For answers to common queries and question regarding Replication within MySQL, see [Section 15.3.4, “Replication FAQ”](#).

A.14. MySQL 5.1 FAQ — MySQL, DRBD, and Heartbeat

A.14.1. Distributed Replicated Block Device

In the following section, we provide answers to questions that are most frequently asked about Distributed Replicated Block Device (DRBD).

Questions

- [30.14.1.1: What is DRBD?](#)
- [30.14.1.2: What are “Block Devices”?](#)
- [30.14.1.3: How is DRBD licensed?](#)
- [30.14.1.4: Where can I download DRBD?](#)
- [30.14.1.5: If I find a bug in DRBD, to whom do I submit the issue?](#)
- [30.14.1.6: Where can I get more technical and business information concerning MySQL and DRBD?](#)

Questions and Answers

30.14.1.1: What is DRBD?

DRBD is an acronym for Distributed Replicated Block Device. DRBD is an open source Linux kernel block device which leverages

synchronous replication to achieve a consistent view of data between two systems, typically an Active and Passive system. DRBD currently supports all the major flavors of Linux and comes bundled in several major Linux distributions. The DRBD project is maintained by [LINBIT](#).

30.14.1.2: What are “Block Devices”?

Block devices are the type of device used to represent storage in the Linux Kernel. All physical disk devices present a “block device” interface. Additionally, virtual disk systems like LVM or DRBD present a “block device” interface. In this way, the file system or other software that might want to access a disk device can be used with any number of real or virtual devices without having to know anything about their underlying implementation details.

30.14.1.3: How is DRBD licensed?

DRBD is licensed under the GPL.

30.14.1.4: Where can I download DRBD?

Please see <http://www.drbd.org/download.html>

30.14.1.5: If I find a bug in DRBD, to whom do I submit the issue?

Bug reports should be submitted to the DRBD mailing list. Please see: <http://lists.linbit.com/> .

30.14.1.6: Where can I get more technical and business information concerning MySQL and DRBD?

Please visit: <http://mysql.com/drbd/>

A.14.2. Linux Heartbeat

In the following section, we provide answers to questions that are most frequently asked about Linux Heartbeat.

Questions

- [30.14.2.1](#): What is Linux Heartbeat?
- [30.14.2.2](#): How is Linux Heartbeat licensed?
- [30.14.2.3](#): Where can I download Linux Heartbeat?
- [30.14.2.4](#): If I find a bug with Linux Heartbeat, to whom do I submit the issue?

Questions and Answers

30.14.2.1: What is Linux Heartbeat?

The Linux-HA project (<http://www.linux-ha.org/>) offers a high availability solution commonly referred to as Linux Heartbeat. Linux Heartbeat ships as part of several Linux distributions, as well as within several embedded high availability systems. This solution can also be used for other applications besides databases servers, such as mail servers, web servers, file servers, and DNS servers.

Linux Heartbeat implements a heartbeat-protocol. A heartbeat-protocol means that messages are sent at regular intervals between two or more nodes. If a message is not received from a node within a given interval, then it is assumed the node has failed and some type of failover or recovery action is required. Linux Heartbeat is typically configured to send these heartbeat messages over standard Ethernet interfaces, but it does also support other methods, such as serial-line links.

30.14.2.2: How is Linux Heartbeat licensed?

Linux Heartbeat is licensed under the GPL.

30.14.2.3: Where can I download Linux Heartbeat?

Please see <http://linux-ha.org/download/index.html>.

30.14.2.4: If I find a bug with Linux Heartbeat, to whom do I submit the issue?

Bug reports should be submitted to <http://www.linux-ha.org/ClusterResourceManager/BugReports>.

A.14.3. DRBD Architecture

In the following section, we provide answers to questions that are most frequently asked about DRBD Architecture.

Questions

- [30.14.3.1](#): Is an Active/Active option available for MySQL with DRBD?
- [30.14.3.2](#): What MySQL storage engines are supported with DRBD?
- [30.14.3.3](#): How long does a failover take?
- [30.14.3.4](#): How long does it take to resynchronize data after a failure?

Questions and Answers

30.14.3.1: Is an Active/Active option available for MySQL with DRBD?

Currently, MySQL does not support Active/Active configurations using DRBD “out of the box”.

30.14.3.2: What MySQL storage engines are supported with DRBD?

All of the MySQL transactional storage engines are supported by DRBD, including InnoDB and Falcon. For archived or read-only data, MyISAM or Archive can also be used.

30.14.3.3: How long does a failover take?

Failover time is dependent on many things, some of which are configurable. After activating the passive host, MySQL will have to start and run a normal recovery process. If the InnoDB log files have been configured to a large size and there was heavy write traffic, this may take a reasonably long period of time. However, under normal circumstances, failover tends to take less than a minute.

30.14.3.4: How long does it take to resynchronize data after a failure?

Resynchronization time depends on how long the two machines are out of communication and how much data was written during that period of time. Resynchronization time is a function of data to be synced, network speed and disk speed. DRBD maintains a bitmap of changed blocks on the primary machine, so only those blocks that have changed will need to be transferred.

A.14.4. DRBD and MySQL Replication

In the following section, we provide answers to questions that are most frequently asked about MySQL Replication Scale-out.

Questions

- [30.14.4.1](#): What is the difference between MySQL Cluster and DRBD?
- [30.14.4.2](#): What is the difference between MySQL Replication and DRBD?
- [30.14.4.3](#): How can I combine MySQL Replication scale-out with DRBD?

Questions and Answers

30.14.4.1: What is the difference between MySQL Cluster and DRBD?

Both MySQL Cluster and DRBD replicate data synchronously. MySQL Cluster leverages a shared-nothing storage architecture in which the cluster can be architected beyond an Active/Passive configuration. DRBD operates at a much lower level within the “stack”, at the disk I/O level. For a comparison of various high availability features between these two options, please refer to [Chapter 14, High Availability, Scalability, and DRBD](#).

30.14.4.2: What is the difference between MySQL Replication and DRBD?

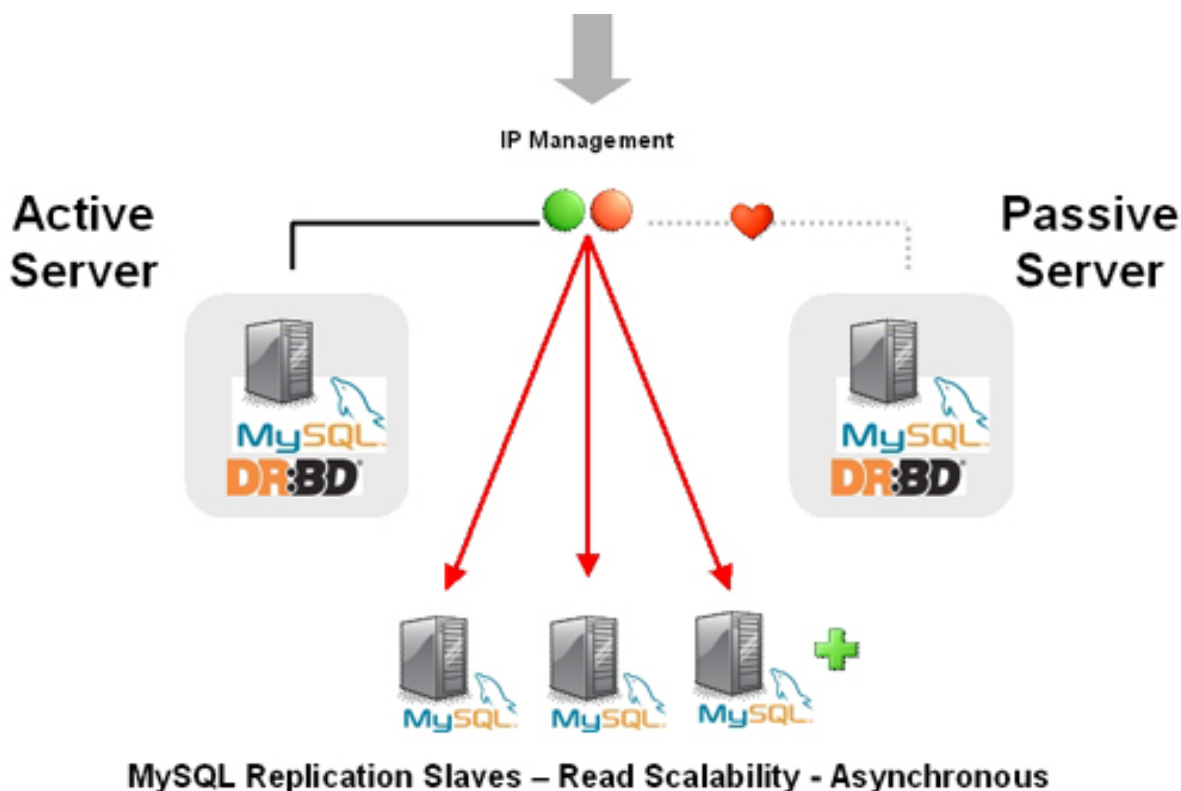
MySQL Replication replicates data asynchronously while DRBD replicates data synchronously. For a comparison of various high availability features between these two options, please refer to the high availability comparison grid, [Chapter 14, High Availability, Scalability, and DRBD](#).

30.14.4.3: How can I combine MySQL Replication scale-out with DRBD?

MySQL Replication is typically deployed in a Master to many Slaves configuration. In this configuration, having many Slaves provides read scalability. DRBD is used to provide high-availability for the Master MySQL Server in an Active/Passive configuration. This provides for automatic failover, safeguards against data loss, and automatically synchronizes the failed MySQL Master after a failover.

The most likely scenario in which MySQL Replication scale-out can be leveraged with DRBD is in the form of attaching replicated MySQL “read-slaves” off of the Active-Master MySQL Server, shown in [Figure A.1, “Active-Master MySQL server”](#). Since DRBD replicates an entire block device, master information such as the binary logs are also replicated. In this way, all of the slaves can attach to the Virtual IP Address managed by Linux Heartbeat. In the event of a failure, the asynchronous nature of MySQL Replication allows the slaves to continue with the new Active machine as their master with no intervention needed.

Figure A.1. Active-Master MySQL server



A.14.5. DRBD and File Systems

In the following section, we provide answers to questions that are most frequently asked about DRBD and file systems.

Questions

- [30.14.5.1: Can XFS be used with DRBD?](#)

Questions and Answers

30.14.5.1: Can XFS be used with DRBD?

Yes. XFS uses dynamic block size, thus DRBD 0.7 or later is needed.

A.14.6. DRBD and LVM

In the following section, we provide answers to questions that are most frequently asked about DRBD and LVM.

Questions

- [30.14.6.1](#): Can I use DRBD on top of LVM?
- [30.14.6.2](#): Can I use LVM on top of DRBD?
- [30.14.6.3](#): Can I use DRBD on top of LVM while at the same time running LVM on top of that DRBD?

Questions and Answers

30.14.6.1: Can I use DRBD on top of LVM?

Yes, DRBD supports on-line resizing. If you enlarge your logical volume that acts as a backing device for DRBD, you can enlarge DRBD itself too, and of course your file system if it supports resizing.

30.14.6.2: Can I use LVM on top of DRBD?

Yes, you can use DRBD as a Physical Volume (PV) for LVM. Depending on the default LVM configuration shipped with your distribution, you may need to add the `/dev/drbd*` device files to the `filter` option in your `lvm.conf` so LVM scans your DRBDs for PV signatures.

30.14.6.3: Can I use DRBD on top of LVM while at the same time running LVM on top of that DRBD?

This requires careful tuning of your LVM configuration to avoid duplicate PV scans, but yes, it is possible.

A.14.7. DRBD and Virtualization

In the following section, we provide answers to questions that are most frequently asked about DRBD and virtualization.

Questions

- [30.14.7.1](#): Can I use DRBD with OpenVZ?
- [30.14.7.2](#): Can I use DRBD with Xen and/or KVM?

Questions and Answers

30.14.7.1: Can I use DRBD with OpenVZ?

See http://wiki.openvz.org/HA_cluster_with_DRBD_and_Heartbeat.

30.14.7.2: Can I use DRBD with Xen and/or KVM?

Yes. If you are looking for professional consultancy or expert commercial support for Xen- or KVM-based virtualization clusters with DRBD, contact LINBIT (<http://www.linbit.com>).

A.14.8. DRBD and Security

In the following section, we provide answers to questions that are most frequently asked about DRBD and security.

Questions

- [30.14.8.1](#): Can I encrypt/compress the exchanged data?

- [30.14.8.2](#): Does DRBD do mutual node authentication?

Questions and Answers

30.14.8.1: Can I encrypt/compress the exchanged data?

Yes. But there is no option within DRBD to allow for this. You'll need to leverage a VPN and the network layer should do the rest.

30.14.8.2: Does DRBD do mutual node authentication?

Yes, starting with DRBD 8 shared-secret mutual node authentication is supported.

A.14.9. DRBD and System Requirements

In the following section, we provide answers to questions that are most frequently asked about DRBD and System Requirements.

Questions

- [30.14.9.1](#): What other packages besides DRBD are required?
- [30.14.9.2](#): How many machines are required to set up DRBD?
- [30.14.9.3](#): Does DRBD only run on Linux?

Questions and Answers

30.14.9.1: What other packages besides DRBD are required?

When using pre-built binary packages, none except a matching kernel, plus packages for [glibc](#) and your favorite shell. When compiling DRBD from source additional prerequisite packages may be required. They include but are not limited to:

- glib-devel
- openssl
- devel
- libgcrypt-devel
- glib2-devel
- pkgconfig
- ncurses-devel
- rpm-build
- rpm-devel
- redhat-rpm-config
- gcc
- gcc-c++
- bison
- flex
- gnutls-devel
- lm_sensors-devel

- net-snmp-devel
- python-devel
- bzip2-devel
- libselenium-devel
- perl-DBI
- libnet

Pre-built x86 and x86_64 packages for specific kernel versions are available with a support subscription from LINBIT. Please note that if the kernel is upgraded, DRBD must be as well.

30.14.9.2: How many machines are required to set up DRBD?

Two machines are required to achieve the minimum degree of high availability. Although at any one given point in time one will be primary and one will be secondary, it is better to consider the machines as part of a mirrored pair without a “natural” primary machine.

30.14.9.3: Does DRBD only run on Linux?

DRBD is a Linux Kernel Module, and can work with many popular Linux distributions. DRBD is currently not available for non-Linux operating systems.

A.14.10. DBRD and Support and Consulting

In the following section, we provide answers to questions that are most frequently asked about DRBD and resources.

Questions

- [30.14.10.1](#): Does MySQL offer professional consulting to help with designing a DRBD system?
- [30.14.10.2](#): Does MySQL offer support for DRBD and Linux Heartbeat from MySQL?
- [30.14.10.3](#): Are pre-built binaries or RPMs available?
- [30.14.10.4](#): Does MySQL have documentation to help me with the installation and configuration of DRBD and Linux Heartbeat?
- [30.14.10.5](#): Is there a dedicated discussion forum for MySQL High-Availability?
- [30.14.10.6](#): Where can I get more information about MySQL for DRBD?

Questions and Answers

30.14.10.1: Does MySQL offer professional consulting to help with designing a DRBD system?

Yes. MySQL offers consulting for the design, installation, configuration, and monitoring of high availability DRBD. For more information concerning a High Availability Jumpstart, please see: <http://www.mysql.com/consulting/packaged/scaleout.html>.

30.14.10.2: Does MySQL offer support for DRBD and Linux Heartbeat from MySQL?

Yes. Support for DRBD and Linux Heartbeat is available with an add-on subscription to MySQL Enterprise called “DRBD for MySQL”. For more information about support options for DRBD see: <http://mysql.com/products/enterprise/features.html>.

For the list of supported Linux distributions, please see: <http://www.mysql.com/support/supportedplatforms/enterprise.html>.

Note

DRBD is only available on Linux. DRBD is not available on Windows, MacOS, Solaris, HP-UX, AIX, FreeBSD, or other non-Linux platforms.

30.14.10.3: Are pre-built binaries or RPMs available?

Yes. “DRBD for MySQL” is an add-on subscription to MySQL Enterprise, which provides pre-built binaries for DRBD. For more information see: <http://mysql.com/products/enterprise/features.html>.

30.14.10.4: Does MySQL have documentation to help me with the installation and configuration of DRBD and Linux Heartbeat?

Coming soon.

30.14.10.5: Is there a dedicated discussion forum for MySQL High-Availability?

Yes, <http://forums.mysql.com/list.php?144>.

30.14.10.6: Where can I get more information about MySQL for DRBD?

For more information about MySQL for DRBD, including a technical white paper please see: [DRBD for MySQL High Availability](#).

Appendix B. Errors, Error Codes, and Common Problems

This appendix lists common problems and errors that may occur and potential resolutions, in addition to listing the errors that may appear when you call MySQL from any host language. The first section covers problems and resolutions. Detailed information on errors is provided; The first list displays server error messages. The second list displays client program messages.

MySQL Enterprise

The MySQL Enterprise Monitor provides a “Virtual DBA” to assist with problem solving. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

B.1. Problems and Common Errors

This section lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

B.1.1. How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:
 - The keyboard doesn't work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light doesn't change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)
 - The mouse pointer doesn't move.
 - The machine doesn't answer to a remote machine's pings.
 - Other programs that are not related to MySQL don't behave correctly.
 - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as [glibc](#)) are up to date.

It's always good to use a machine with ECC memory to discover memory problems early.

- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See [Section 5.11, “MySQL Server Logs”](#).
- If you don't think you have hardware problems, you should try to find out which program is causing problems. Try using `top`, `ps`, Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use `top`, `df`, or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it doesn't want to die, there is probably a bug in the operating system.

If after you have examined all other possibilities and you have concluded that the MySQL server or a MySQL client is causing the problem, it's time to create a bug report for our mailing list or our support team. In the bug report, try to give a very detailed description of how the system is behaving and what you think is happening. You should also state why you think that MySQL is causing the problem. Take into consideration all the situations in this chapter. State any problems exactly how they appear when you examine your system. Use the “copy and paste” method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only “the system doesn't work.” This doesn't provide us with any information about what could be the problem.

If a program fails, it's always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with `top`. Let the program run for a while, it may simply be evaluating something computationally intensive.
- If the `mysqld` server is causing problems, can you get any response from it with `mysqladmin -u root ping` or `mysqladmin -u root processlist`?
- What does a client program say when you try to connect to the MySQL server? (Try with `mysql`, for example.) Does the client jam? Do you get any output from the program?

When sending a bug report, you should follow the outline described in [Section 1.8, “How to Report Bugs or Problems”](#).

B.1.2. Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

B.1.2.1. Access denied

An `Access denied` error can have many causes. Often the problem is related to the MySQL accounts that the server allows client programs to use when connecting. See [Section 5.7.8, “Causes of Access denied Errors”](#), and [Section 5.7.2, “How the Privilege System Works”](#).

B.1.2.2. Can't connect to [local] MySQL server

A MySQL client on Unix can connect to the `mysqld` server in two different ways: By using a Unix socket file to connect through a file in the filesystem (default `/tmp/mysql.sock`), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you don't specify a hostname or if you specify the special hostname `localhost`.

If the MySQL server is running on Windows, you can connect via TCP/IP. If the server is started with the `--enable-named-pipe` option, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is `MySQL` by default. If you don't give a hostname when connecting to `mysqld`, a MySQL client first tries to connect to the named pipe. If that doesn't work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using `.` as the hostname.

The error (2002) `Can't connect to ...` normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket filename or TCP/IP port number when trying to connect to the server.

The error (2003) `Can't connect to MySQL server on 'server' (10061)` indicates that the network connection has been refused. You should check that there is a MySQL server running, that it has network connections enabled, the network port you specified is the one configured on the server, and that the TCP/IP port you are using has not been blocked by a firewall or port blocking service.

Start by checking whether there is a process named `mysqld` running on your server host. (Use `ps xa | grep mysqld` on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See [Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”](#).

If a `mysqld` process is running, you can check it by trying the following commands. The port number or Unix socket filename might be different in your setup. `host_ip` represents the IP number of the machine where the server is running.

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h `hostname` version variables
shell> mysqladmin -h `hostname` --port=3306 version
shell> mysqladmin -h host_ip version
shell> mysqladmin --protocol=socket --socket=/tmp/mysql.sock version
```

Note the use of backticks rather than forward quotes with the `hostname` command; these cause the output of `hostname` (that is, the current hostname) to be substituted into the `mysqladmin` command. If you have no `hostname` command or are running on Windows, you can manually type the hostname of your machine (without backticks) following the `-h` option. You can also try `-h 127.0.0.1` to connect with TCP/IP to the local host.

Here are some reasons the `Can't connect to local MySQL server` error might occur:

- `mysqld` is not running. Check your operating system's process list to ensure the `mysqld` process is present.
- You're running a MySQL server on Windows with many TCP/IP connections to it. If you're experiencing that quite often your clients get that error, you can find a workaround here: [Section B.1.2.2.1, "Connection to MySQL Server Failing on Windows"](#).
- You are running on a system that uses MIT-pthreads. If you are running on a system that doesn't have native threads, `mysqld` uses the MIT-pthreads package. See [Section 2.1.1, "Operating Systems Supported by MySQL Community Server"](#). However, not all MIT-pthreads versions support Unix socket files. On a system without socket file support, you must always specify the hostname explicitly when connecting to the server. Try using this command to check the connection to the server:

```
shell> mysqladmin -h `hostname` version
```

- Someone has removed the Unix socket file that `mysqld` uses (`/tmp/mysql.sock` by default). For example, you might have a `cron` job that removes old files from the `/tmp` directory. You can always run `mysqladmin version` to check whether the Unix socket file that `mysqladmin` is trying to use really exists. The fix in this case is to change the `cron` job to not remove `mysql.sock` or to place the socket file somewhere else. See [Section B.1.4.5, "How to Protect or Change the MySQL Unix Socket File"](#).
- You have started the `mysqld` server with the `--socket=/path/to/socket` option, but forgotten to tell client programs the new name of the socket file. If you change the socket pathname for the server, you must also notify the MySQL clients. You can do this by providing the same `--socket` option when you run client programs. You also need to ensure that clients have permission to access the `mysql.sock` file. To find out where the socket file is, you can do:

```
shell> netstat -ln | grep mysql
```

See [Section B.1.4.5, "How to Protect or Change the MySQL Unix Socket File"](#).

- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other `mysqld` threads (for example, with `kill` or with the `mysql_zap` script) before you can restart the MySQL server. See [Section B.1.4.2, "What to Do If MySQL Keeps Crashing"](#).
- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or socket file so that the server and clients can access them, or restart `mysqld` with a `--socket` option that specifies a socket filename in a directory where the server can create it and where client programs can access it.

If you get the error message `Can't connect to MySQL server on some_host`, you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing `telnet some_host 3306` and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.
- If the server is running on the local host, try using `mysqladmin -h localhost variables` to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the `port` variable.)
- Make sure that your `mysqld` server was not started with the `--skip-networking` option. If it was, you cannot connect to it using TCP/IP.
- Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default).

Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked.

Under Windows, applications such as ZoneAlarm and the Windows XP personal firewall may need to be configured to allow external access to a MySQL server.

- If you are running under Linux and Security-Enhanced Linux (SELinux) is enabled, make sure you have disabled SELinux protection for the `mysqld` process.

B.1.2.2.1. Connection to MySQL Server Failing on Windows

When you're running a MySQL server on Windows with many TCP/IP connections to it, and you're experiencing that quite often your clients get a `Can't connect to MySQL server` error, the reason might be that Windows doesn't allow for enough ephemeral (short-lived) ports to serve those connections.

By default, Windows allows 5000 ephemeral (short-lived) TCP ports to the user. After any port is closed it will remain in a `TIME_WAIT` status for 120 seconds. This status allows the connection to be reused at a much lower cost than reinitializing a brand new connection. However, the port will not be available again until this time expires.

With a small stack of available TCP ports (5000) and a high number of TCP ports being open and closed over a short period of time along with the `TIME_WAIT` status you have a good chance for running out of ports. There are two ways to address this problem:

- Reduce the number of TCP ports consumed quickly by investigating connection pooling or persistent connections where possible
- Tune some settings in the Windows registry (see below)

IMPORTANT: The following procedure involves modifying the Windows registry. Before you modify the registry, make sure to back it up and make sure that you understand how to restore the registry if a problem occurs. For information about how to back up, restore, and edit the registry, view the following article in the Microsoft Knowledge Base: <http://support.microsoft.com/kb/256986/EN-US/>.

1. Start Registry Editor (`Regedt32.exe`).
2. Locate the following key in the registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

3. On the `Edit` menu, click `Add Value`, and then add the following registry value:

```
Value Name: MaxUserPort
Data Type: REG_DWORD
Value: 65534
```

This sets the number of ephemeral ports available to any user. The valid range is between 5000 and 65534 (decimal). The default value is 0x1388 (5000 decimal).

4. On the `Edit` menu, click `Add Value`, and then add the following registry value:

```
Value Name: TcpTimedWaitDelay
Data Type: REG_DWORD
Value: 30
```

This sets the number of seconds to hold a TCP port connection in `TIME_WAIT` state before closing. The valid range is between 0 (zero) and 300 (decimal). The default value is 0x78 (120 decimal).

5. Quit Registry Editor.
6. Reboot the machine.

Note: Undoing the above should be as simple as deleting the registry entries you've created.

B.1.2.3. Client does not support authentication protocol

MySQL 5.1 uses an authentication protocol based on a password hashing algorithm that is incompatible with that used by older (pre-4.1) clients. If you upgrade the server from 4.0, attempts to connect to it with an older client may fail with the following message:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

To solve this problem, you should use one of the following approaches:

- Upgrade all client programs to use a 4.1.1 or newer client library.
- When connecting to the server with a pre-4.1 client program, use an account that still has a pre-4.1-style password.
- Reset the password to pre-4.1 style for each user that needs to use a pre-4.1 client program. This can be done using the `SET PASSWORD` statement and the `OLD_PASSWORD( )` function:

```
mysql> SET PASSWORD FOR
-> 'some_user'@'some_host' = OLD_PASSWORD('newpwd');
```

Alternatively, use `UPDATE` and `FLUSH PRIVILEGES`:

```
mysql> UPDATE mysql.user SET Password = OLD_PASSWORD('newpwd')
-> WHERE Host = 'some_host' AND User = 'some_user';
mysql> FLUSH PRIVILEGES;
```

Substitute the password you want to use for “*newpwd*” in the preceding examples. MySQL cannot tell you what the original password was, so you'll need to pick a new one.

- Tell the server to use the older password hashing algorithm:
 1. Start `mysqld` with the `--old-passwords` option.
 2. Assign an old-format password to each account that has had its password updated to the longer 4.1 format. You can identify these accounts with the following query:

```
mysql> SELECT Host, User, Password FROM mysql.user
-> WHERE LENGTH>Password) > 16;
```

For each account record displayed by the query, use the `Host` and `User` values and assign a password using the `OLD_PASSWORD( )` function and either `SET PASSWORD` or `UPDATE`, as described earlier.

Note

In older versions of PHP, the `mysql` extension does not support the authentication protocol in MySQL 4.1.1 and higher. This is true regardless of the PHP version being used. If you wish to use the `mysql` extension with MySQL 4.1 or newer, you may need to follow one of the options discussed above for configuring MySQL to work with old clients. The `mysqli` extension (stands for “MySQL, Improved”; added in PHP 5) is compatible with the improved password hashing employed in MySQL 4.1 and higher, and no special configuration of MySQL need be done to use this MySQL client library. For more information about the `mysqli` extension, see <http://php.net/mysqli>.

It may also be possible to compile the older `mysql` extension against the new MySQL client library. This is beyond the scope of this Manual; consult the PHP documentation for more information. You also be able to obtain assistance with these issues in our [MySQL with PHP forum](#).

For additional background on password hashing and authentication, see [Section 5.7.9, “Password Hashing as of MySQL 4.1”](#).

B.1.2.4. Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a `--password` or `-p` option that has no following password value:

```
shell> mysql -u user_name -p
```

```
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the `Enter password:` prompt. This occurs when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

B.1.2.5. Host '*host_name*' is blocked

If you get the following error, it means that `mysqld` has received many connect requests from the host '*host_name*' that have been interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

The number of interrupted connect requests allowed is determined by the value of the `max_connect_errors` system variable. After `max_connect_errors` failed requests, `mysqld` assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connections until you execute a `mysqladmin flush-hosts` command or issue a `FLUSH HOSTS` statement. See [Section 5.2.3, “System Variables”](#).

By default, `mysqld` blocks a host after 10 connection errors. You can adjust the value by starting the server like this:

```
shell> mysqld_safe --max_connect_errors=10000 &
```

If you get this error message for a given host, you should first verify that there isn't anything wrong with TCP/IP connections from that host. If you are having network problems, it does you no good to increase the value of the `max_connect_errors` variable.

B.1.2.6. Too many connections

If you get a `Too many connections` error when you try to connect to the `mysqld` server, this means that all available connections are in use by other clients.

The number of connections allowed is controlled by the `max_connections` system variable. Beginning with MySQL 5.1.15, its default value is 151 to improve performance when MySQL is used with the Apache Web server. (Previously, the default was 100.) If you need to support more connections, you should restart `mysqld` with a larger value for this variable.

MySQL Enterprise

Subscribers to the MySQL Enterprise Monitor receive advice on dynamically configuring the `max_connections` variable — avoiding failed connection attempts. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

`mysqld` actually allows `max_connections+1` clients to connect. The extra connection is reserved for use by accounts that have the `SUPER` privilege. By granting the `SUPER` privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See [Section 12.5.4.25, “SHOW PROCESSLIST Syntax”](#).

The maximum number of connections MySQL can support depends on the quality of the thread library on a given platform. Linux or Solaris should be able to support 500-1000 simultaneous connections, depending on how much RAM you have and what your clients are doing. Static Linux binaries provided by MySQL AB can support up to 4000 connections.

B.1.2.7. Out of memory

If you issue a query using the `mysql` client program and receive an error like the following one, it means that `mysql` does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke `mysql` with the `--quick` option. This causes it to use the `mysql_use_result()` C API function to retrieve the result set, which places less of a load on the client (but more on the server).

B.1.2.8. MySQL server has gone away

This section also covers the related [Lost connection to server during query](#) error.

The most common reason for the [MySQL server has gone away](#) error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent):

Error Code	Description
CR_SERVER_GONE_ERROR	The client couldn't send a question to the server.
CR_SERVER_LOST	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the [wait_timeout](#) variable when you start `mysqld`. See [Section 5.2.3, “System Variables”](#).

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the `mysql` command-line client).

Some other common reasons for the [MySQL server has gone away](#) error are:

- You (or the db administrator) has killed the running thread with a [KILL](#) statement or a `mysqladmin kill` command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- A client application running on a different host does not have the necessary privileges to connect to the MySQL server from that host.
- You got a timeout from the TCP/IP connection on the client side. This may happen if you have been using the commands: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT, ...)` or `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT, ...)`. In this case increasing the timeout may help solve the problem.
- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the [reconnect](#) flag in the `MYSQL` structure is equal to 0).
- You are using a Windows client and the server had dropped the connection (probably because [wait_timeout](#) expired) before the command was issued.

The problem on Windows is that in some cases MySQL doesn't get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from the connection.

Prior to MySQL 5.1.8, even if the [reconnect](#) flag in the `MYSQL` structure is equal to 1, MySQL does not automatically reconnect and re-issue the query as it doesn't know if the server did get the original query or not.

The solution to this is to either do a [mysql_ping](#) on the connection if there has been a long time since the last query (this is what `MyODBC` does) or set [wait_timeout](#) on the `mysqld` server so high that it in practice never times out.

- You can also get these errors if you send a query to the server that is incorrect or too large. If `mysqld` receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big [BLOB](#) columns), you can increase the query limit by setting the server's [max_allowed_packet](#) variable, which has a default value of 1MB. You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in [Section B.1.2.9, “Packet too large”](#).

An [INSERT](#) or [REPLACE](#) statement that inserts a great many rows can also cause these sorts of errors. Either one of these statements sends a single request to the server irrespective of the number of rows to be inserted; thus, you can often avoid the error by reducing the number of rows sent per [INSERT](#) or [REPLACE](#).

- You also get a lost connection if you are sending a packet 16MB or larger if your client is older than 4.0.8 and your server is 4.0.8 and above, or the other way around.
- It is also possible to see this error if hostname lookups fail (for example, if the DNS server on which your server or network relies goes down). This is because MySQL is dependent on the host system for name resolution, but has no way of knowing whether it is working — from MySQL's point of view the problem is indistinguishable from any other network timeout.

You may also see the [MySQL server has gone away](#) error if MySQL is started with the `--skip-networking` option.

Another networking issue that can cause this error occurs if the MySQL port (default 3306) is blocked by your firewall, thus preventing any connections at all to the MySQL server.

- You can also encounter this error with applications that fork child processes, all of which try to use the same connection to the MySQL server. This can be avoided by using a separate connection for each child process.
- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing `mysqladmin version` and examining the server's up-time. If the client connection was broken because `mysqld` crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See [Section B.1.4.2, “What to Do If MySQL Keeps Crashing”](#).

You can get more information about the lost connections by starting `mysqld` with the `--log-warnings=2` option. This logs some of the disconnected errors in the `hostname.err` file. See [Section 5.11.2, “The Error Log”](#).

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether the MySQL server died. You can find information about this in the server error log. See [Section B.1.4.2, “What to Do If MySQL Keeps Crashing”](#).
- If a specific query kills `mysqld` and the tables involved were checked with `CHECK TABLE` before you ran the query, can you provide a reproducible test case? See [MySQL Internals: Porting](#).
- What is the value of the `wait_timeout` system variable in the MySQL server? (`mysqladmin variables` gives you the value of this variable.)
- Have you tried to run `mysqld` with the `--log` option to determine whether the problem query appears in the log?

See also [Section B.1.2.10, “Communication Errors and Aborted Connections”](#), and [Section 1.8, “How to Report Bugs or Problems”](#).

B.1.2.9. Packet too large

A communication packet is a single SQL statement sent to the MySQL server, a single row that is sent to the client, or a binary log event sent from a master replication server to a slave.

The largest possible packet that can be transmitted to or from a MySQL 5.1 server or client is 1GB.

When a MySQL client or the `mysqld` server receives a packet bigger than `max_allowed_packet` bytes, it issues a **Packet too large** error and closes the connection. With some clients, you may also get a **Lost connection to MySQL server during query** error if the communication packet is too large.

Both the client and the server have their own `max_allowed_packet` variable, so if you want to handle big packets, you must increase this variable both in the client and in the server.

If you are using the `mysql` client program, its default `max_allowed_packet` variable is 16MB. To set a larger value, start `mysql` like this:

```
shell> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default `max_allowed_packet` value is 1MB. You can increase this if the server needs to handle big queries (for example, if you are working with big **BLOB** columns). For example, to set the variable to 16MB, start the server like this:

```
shell> mysqld --max_allowed_packet=16M
```

You can also use an option file to set `max_allowed_packet`. For example, to set the size for the server to 16MB, add the following lines in an option file:

```
[mysqld]  
max_allowed_packet=16M
```

It is safe to increase the value of this variable because the extra memory is allocated only when needed. For example, `mysqld` allocates more memory only when you issue a long query or when `mysqld` must return a large result row. The small default value of the variable is a precaution to catch incorrect packets between the client and server and also to ensure that you do not run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large `BLOB` values but have not given `mysqld` access to enough memory to handle the query. If you suspect this is the case, try adding `ulimit -d 256000` to the beginning of the `mysqld_safe` script and restarting `mysqld`.

B.1.2.10. Communication Errors and Aborted Connections

The server error log can be a useful source of information about connection problems. See [Section 5.11.2, “The Error Log”](#). If you start the server with the `--log-warnings` option, you might find messages like this in your error log:

```
010301 14:38:23 Aborted connection 854 to db: 'users' user: 'josh'
```

If `Aborted connections` messages appear in the error log, the cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout` or `interactive_timeout` seconds without issuing any requests to the server. See [Section 5.2.3, “System Variables”](#).
- The client program ended abruptly in the middle of a data transfer.

When any of these things happen, the server increments the `Aborted_clients` status variable.

The server increments the `Aborted_connects` status variable when the following things happen:

- A client doesn't have privileges to connect to a database.
- A client uses an incorrect password.
- A connection packet doesn't contain the right information.
- It takes more than `connect_timeout` seconds to get a connect packet. See [Section 5.2.3, “System Variables”](#).

If these kinds of things happen, it might indicate that someone is trying to break into your server!

MySQL Enterprise

For reasons of security and performance the advisors provided by the MySQL Enterprise Monitor pay special attention to the `Aborted_connections` status variable. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

Other reasons for problems with aborted clients or aborted connections:

- Use of Ethernet protocol with Linux, both half and full duplex. Many Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file via FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. The only solution is switching the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and testing the results to determine the best setting.
- Some problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.
- The `max_allowed_packet` variable value is too small or queries require more memory than you have allocated for `mysqld`. See [Section B.1.2.9, “Packet too large”](#).

See also [Section B.1.2.8, “MySQL server has gone away”](#).

B.1.2.11. The table is full

The maximum effective table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. The following table lists some examples of operating system file-size limits. This is only a rough guide and is not intended to be definitive. For the most up-to-date information, be sure to check the documentation specific to your operating system.

Operating System	File-size Limit
Win32 w/ FAT/FAT32	2GB/4GB
Win32 w/ NTFS	2TB (possibly larger)
Linux 2.2-Intel 32-bit	2GB (LFS: 4GB)
Linux 2.4+	(using ext3 filesystem) 4TB
Solaris 9/10	16TB
MacOS X w/ HFS+	2TB
NetWare w/NSS filesystem	8TB

Windows users, please note that FAT and VFAT (FAT32) are *not* considered suitable for production use with MySQL. Use NTFS instead.

On Linux 2.2, you can get [MyISAM](#) tables larger than 2GB in size by using the Large File Support (LFS) patch for the ext2 filesystem. Most current Linux distributions are based on kernel 2.4 or higher and include all the required LFS patches. On Linux 2.4, patches also exist for ReiserFS to get support for big files (up to 2TB). With JFS and XFS, petabyte and larger files are possible on Linux.

For a detailed overview about LFS in Linux, have a look at Andreas Jaeger's *Large File Support in Linux* page at http://www.suse.de/~aj/linux_lfs.html.

If you do encounter a full-table error, there are several reasons why it might have occurred:

- The [InnoDB](#) storage engine maintains [InnoDB](#) tables within a tablespace that can be created from several files. This allows a table to exceed the maximum individual file size. The tablespace can include raw disk partitions, which allows extremely large tables. The maximum tablespace size is 64TB.

If you are using [InnoDB](#) tables and run out of room in the [InnoDB](#) tablespace. In this case, the solution is to extend the [InnoDB](#) tablespace. See [Section 13.5.7, “Adding and Removing InnoDB Data and Log Files”](#).

- You are using [MyISAM](#) tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.
- You are using a [MyISAM](#) table and the space required for the table exceeds what is allowed by the internal pointer size. [MyISAM](#) creates tables to allow up to 256GB by default, but this limit can be changed up to the maximum allowable size of 65,536TB ($256^7 - 1$ bytes).

If you need a [MyISAM](#) table that is larger than the default limit and your operating system supports large files, the [CREATE TABLE](#) statement supports [AVG_ROW_LENGTH](#) and [MAX_ROWS](#) options. See [Section 12.1.10, “CREATE TABLE Syntax”](#). The server uses these options to determine how large a table to allow.

If the pointer size is too small for an existing table, you can change the options with [ALTER TABLE](#) to increase a table's maximum allowable size. See [Section 12.1.4, “ALTER TABLE Syntax”](#).

```
ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
```

You have to specify [AVG_ROW_LENGTH](#) only for tables with [BLOB](#) or [TEXT](#) columns; in this case, MySQL can't optimize the space required based only on the number of rows.

To change the default size limit for [MyISAM](#) tables, set the [myisam_data_pointer_size](#), which sets the number of bytes used for internal row pointers. The value is used to set the pointer size for new tables if you do not specify the [MAX_ROWS](#) option. The value of [myisam_data_pointer_size](#) can be from 2 to 7. A value of 4 allows tables up to 4GB; a value of 6 allows tables up to 256TB.

You can check the maximum data and index sizes by using this statement:

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
```

You also can use `myisamchk -dv /path/to/table-index-file`. See [Section 12.5.4, “SHOW Syntax”](#), or [Section 7.5, “myisamchk — MyISAM Table-Maintenance Utility”](#).

Other ways to work around file-size limits for **MyISAM** tables are as follows:

- If your large table is read-only, you can use `myisampack` to compress it. `myisampack` usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. `myisampack` also can merge multiple tables into a single table. See [Section 7.7, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#).
- MySQL includes a **MERGE** library that allows you to handle a collection of **MyISAM** tables that have identical structure as a single **MERGE** table. See [Section 13.6, “The MERGE Storage Engine”](#).
- You are using the **NDB** storage engine, in which case you need to increase the values for the **DataMemory** and **IndexMemory** configuration parameters in your `config.ini` file. See [Section 16.4.5.1, “Data Node Configuration Parameters”](#).
- You are using the **MEMORY (HEAP)** storage engine; in this case you need to increase the value of the `max_heap_table_size` system variable. See [Section 5.2.3, “System Variables”](#).

B.1.2.12. Can't create/write to file

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file '...\sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start `mysqld` with the `--tmpdir` option or to add the option to the `[mysqld]` section of your option file. For example, to specify a directory of `C:\temp`, use these lines:

```
[mysqld]  
tmpdir=C:/temp
```

The `C:\temp` directory must exist and have sufficient space for the MySQL server to write to. See [Section 4.3.2, “Using Option Files”](#).

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the `tmpdir` directory.

Check also the error code that you get with `pererror`. One reason the server cannot write to a table is that the filesystem is full:

```
shell> pererror 28  
OS error code 28: No space left on device
```

If you get an error of the following type during startup, it indicates that the filesystem and/or directory used for storing data files is write protected. Providing the write error is to a test file, This error is not serious and can be safely ignored.

```
Can't create test file /usr/local/mysql/data/master.lower-test
```

B.1.2.13. Commands out of sync

If you get **Commands out of sync; you can't run this command now** in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without calling `mysql_use_result()` or `mysql_store_result()` in between.

B.1.2.14. Ignoring user

If you get the following error, it means that when `mysql` was started or when it reloaded the grant tables, it found an account in the `user` table that had an invalid password.

Found wrong password for user 'some_user'@'some_host'; ignoring user

As a result, the account is simply ignored by the permission system.

The following list indicates possible causes of and fixes for this problem:

- You may be running a new version of `mysql` with an old `user` table. You can check this by executing `mysqlshow mysql user` to see whether the `Password` column is shorter than 16 characters. If so, you can correct this condition by running the `scripts/add_long_password` script.
- The account has an old password (eight characters long) and you didn't start `mysql` with the `--old-protocol` option. Update the account in the `user` table to have a new password or restart `mysql` with the `--old-protocol` option.
- You have specified a password in the `user` table without using the `PASSWORD()` function. Use `mysql` to update the account in the `user` table with a new password, making sure to use the `PASSWORD()` function:

```
mysql> UPDATE user SET Password=PASSWORD('newpwd')
-> WHERE User='some_user' AND Host='some_host';
```

B.1.2.15. Table 'tbl_name' doesn't exist

If you get either of the following errors, it usually means that no table exists in the default database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case sensitive if they are located on a filesystem that has case-sensitive filenames.
- Even for filesystems that are not case sensitive, such as on Windows, all references to a given table within a query must use the same lettercase.

You can check which tables are in the default database with `SHOW TABLES`. See [Section 12.5.4, “SHOW Syntax”](#).

B.1.2.16. Can't initialize character set

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

This error can have any of the following causes:

- The character set is a multi-byte character set and you have no support for the character set in the client. In this case, you need to re-compile the client by running `configure` with the `--with-charset=charset_name` or `--with-extra-charsets=charset_name` option. See [Section 2.9.2, “Typical configure Options”](#).

All standard MySQL binaries are compiled with `--with-extra-character-sets=complex`, which enables support for all multi-byte character sets. See [Section 5.10.1, “The Character Set Used for Data and Sorting”](#).

- The character set is a simple character set that is not compiled into `mysql`, and the character set definition files are not in the place where the client expects to find them.

In this case, you need to use one of the following methods to solve the problem:

- Recompile the client with support for the character set. See [Section 2.9.2, “Typical configure Options”](#).
- Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.
- Copy the character definition files to the path where the client expects them to be.

B.1.2.17. '**FILE**' NOT FOUND and Similar Errors

If you get `ERROR '...' not found (errno: 23), Can't open file: ... (errno: 24)`, or any other error with `errno 23` or `errno 24` from MySQL, it means that you haven't allocated enough file descriptors for the MySQL server. You can use the `pererror` utility to get a description of what the error number means:

```
shell> pererror 23
OS error code 23: File table overflow
shell> pererror 24
OS error code 24: Too many open files
shell> pererror 11
OS error code 11: Resource temporarily unavailable
```

The problem here is that `mysqld` is trying to keep open too many files simultaneously. You can either tell `mysqld` not to open so many files at once or increase the number of file descriptors available to `mysqld`.

To tell `mysqld` to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the `table_open_cache` system variable (the default value is 64). Reducing the value of `max_connections` also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to `mysqld`, you can use the `--open-files-limit` option to `mysqld_safe` or (as of MySQL 3.23.30) set the `open_files_limit` system variable. See [Section 5.2.3, “System Variables”](#). The easiest way to set these values is to add an option to your option file. See [Section 4.3.2, “Using Option Files”](#). If you have an old version of `mysqld` that doesn't support setting the open files limit, you can edit the `mysqld_safe` script. There is a commented-out line `ulimit -n 256` in the script. You can remove the `#` character to uncomment this line, and change the number `256` to set the number of file descriptors to be made available to `mysqld`.

`--open-files-limit` and `ulimit` can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a “hard” limit that can be overridden only if you start `mysqld_safe` or `mysqld` as `root` (just remember that you also need to start the server with the `--user` option in this case so that it does not continue to run as `root` after it starts up). If you need to increase the operating system limit on the number of file descriptors available to each process, consult the documentation for your system.

Note

If you run the `tcsh` shell, `ulimit` does not work! `tcsh` also reports incorrect values when you ask for the current limits. In this case, you should start `mysqld_safe` using `sh`.

B.1.3. Installation-Related Issues

B.1.3.1. Problems Linking to the MySQL Client Library

When you are linking an application program to use the MySQL client library, you might get undefined reference errors for symbols that start with `mysql_`, such as those shown here:

```
/tmp/ccFKsdPa.o: In function `main':
/tmp/ccFKsdPa.o(.text+0xb): undefined reference to `mysql_init'
/tmp/ccFKsdPa.o(.text+0x31): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x57): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x69): undefined reference to `mysql_error'
/tmp/ccFKsdPa.o(.text+0x9a): undefined reference to `mysql_close'
```

You should be able to solve this problem by adding `-Ldir_path -lmysqlclient` at the end of your link command, where `dir_path` represents the pathname of the directory where the client library is located. To determine the correct directory, try this command:

```
shell> mysql_config --libs
```

The output from `mysql_config` might indicate other libraries that should be specified on the link command as well.

If you get `undefined reference` errors for the `uncompress` or `compress` function, add `-lz` to the end of your link command and try again.

If you get `undefined reference` errors for a function that should exist on your system, such as `connect`, check the manual page for the function in question to determine which libraries you should add to the link command.

You might get `undefined reference` errors such as the following for functions that don't exist on your system:

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

This usually means that your MySQL client library was compiled on a system that is not 100% compatible with yours. In this case, you should download the latest MySQL source distribution and compile MySQL yourself. See [Section 2.9, “MySQL Installation Using a Source Distribution”](#).

You might get undefined reference errors at runtime when you try to execute a MySQL program. If these errors specify symbols that start with `mysql_` or indicate that the `mysqlclient` library can't be found, it means that your system can't find the shared `libmysqlclient.so` library. The fix for this is to tell your system to search for shared libraries where the library is located. Use whichever of the following methods is appropriate for your system:

- Add the path to the directory where `libmysqlclient.so` is located to the `LD_LIBRARY_PATH` environment variable.
- Add the path to the directory where `libmysqlclient.so` is located to the `LD_LIBRARY` environment variable.
- Copy `libmysqlclient.so` to some directory that is searched by your system, such as `/lib`, and update the shared library information by executing `ldconfig`.

Another way to solve this problem is by linking your program statically with the `-static` option, or by removing the dynamic MySQL libraries before linking your code. Before trying the second method, you should be sure that no other programs are using the dynamic libraries.

B.1.3.2. Problems with File Permissions

If you have problems with file permissions, the `UMASK` environment variable might be set incorrectly when `mysqld` starts. For example, MySQL might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

The default `UMASK` value is `0660`. You can change this behavior by starting `mysqld_safe` as follows:

```
shell> UMASK=384 # = 600 in octal
shell> export UMASK
shell> mysqld_safe &
```

By default, MySQL creates database directories with an access permission value of `0700`. You can modify this behavior by setting the `UMASK_DIR` variable. If you set its value, new directories are created with the combined `UMASK` and `UMASK_DIR` values. For example, if you want to give group access to all new directories, you can do this:

```
shell> UMASK_DIR=504 # = 770 in octal
shell> export UMASK_DIR
shell> mysqld_safe &
```

In MySQL 3.23.25 and above, MySQL assumes that the value for `UMASK` and `UMASK_DIR` is in octal if it starts with a zero.

See [Section 2.14, “Environment Variables”](#).

B.1.4. Administration-Related Issues

B.1.4.1. How to Reset the Root Password

If you have never set a `root` password for MySQL, the server does not require a password at all for connecting as `root`. However, it

is recommended to set a password for each account. See [Section 5.6.1, “General Security Guidelines”](#).

If you set a `root` password previously, but have forgotten what it was, you can set a new password. The following procedure is for Windows systems. The procedure for Unix systems is given later in this section.

The procedure under Windows:

1. Log on to your system as Administrator.
2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager:

```
Start Menu -> Control Panel -> Administrative Tools -> Services
```

Then find the MySQL service in the list, and stop it.

If your server is not running as a service, you may need to use the Task Manager to force it to stop.

3. Create a text file and place the following command within it on a single line:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPassword');
```

Save the file with any name. For this example the file will be `C:\mysql-init.txt`.

4. Open a console window to get to the DOS command prompt:

```
Start Menu -> Run -> cmd
```

5. We are assuming that you installed MySQL to `C:\mysql`. If you installed MySQL to another location, adjust the following commands accordingly.

At the DOS command prompt, execute this command:

```
C:\> C:\mysql\bin\mysqld-nt --init-file=C:\mysql-init.txt
```

The contents of the file named by the `--init-file` option are executed at server startup, changing the `root` password. After the server has started successfully, you should delete `C:\mysql-init.txt`.

If you install MySQL using the MySQL Installation Wizard, you may need to specify a `--defaults-file` option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld-nt.exe"  
--defaults-file="C:\Program Files\MySQL\MySQL Server 5.1\my.ini"  
--init-file=C:\mysql-init.txt
```

The appropriate `--defaults-file` setting can be found using the Services Manager:

```
Start Menu -> Control Panel -> Administrative Tools -> Services
```

Find the MySQL service in the list, right-click on it, and choose the [Properties](#) option. The [Path to executable](#) field contains the `--defaults-file` setting.

6. Stop the MySQL server, then restart it in normal mode again. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.
7. You should be able to connect using the new password.

In a Unix environment, the procedure for resetting the `root` password is as follows:

MySQL Enterprise

For expert advice on security-related issues, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

1. Log on to your system as either the Unix `root` user or as the same user that the `mysqld` server runs as.

2. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, hostname, and configuration. Common locations are `/var/lib/mysql/`, `/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the filename has the extension of `.pid` and begins with either `mysqld` or your system's hostname.

You can stop the MySQL server by sending a normal `kill` (not `kill -9`) to the `mysqld` process, using the pathname of the `.pid` file in the following command:

```
shell> kill `cat /mysql-data-directory/host_name.pid`
```

Note the use of backticks rather than forward quotes with the `cat` command; these cause the output of `cat` to be substituted into the `kill` command.

3. Create a text file and place the following command within it on a single line:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPassword');
```

Save the file with any name. For this example the file will be `~/mysql-init`.

4. Restart the MySQL server with the special `--init-file=~/mysql-init` option:

```
shell> mysqld_safe --init-file=~/mysql-init &
```

The contents of the init-file are executed at server startup, changing the root password. After the server has started successfully you should delete `~/mysql-init`.

5. You should be able to connect using the new password.

Alternatively, on any platform, you can set the new password using the `mysql` client (but this approach is less secure):

1. Stop `mysqld` and restart it with the `--skip-grant-tables --user=root` options (Windows users omit the `--user=root` portion).
2. Connect to the `mysqld` server with this command:

```
shell> mysql -u root
```

3. Issue the following statements in the `mysql` client:

```
mysql> UPDATE mysql.user SET Password=PASSWORD('newpwd')  
-> WHERE User='root';  
mysql> FLUSH PRIVILEGES;
```

Replace “*newpwd*” with the actual `root` password that you want to use.

4. You should be able to connect using the new password.

B.1.4.2. What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This doesn't mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the `mysqld` server dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died and restarted, you may find the reason by looking in the server's error log. See [Section 5.11.2, “The Error Log”](#).

On some systems, you can find in the error log a stack trace of where `mysqld` died that you can resolve with the `re-solve_stack_dump` program. See [MySQL Internals: Porting](#). Note that the variable values written in the error log may not always be 100% correct.

Many server crashes are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with `-delay-key-write`, in which case data files are written but not index files.) This means that data file contents are safe even if `mysqld` crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting `mysqld` with the `--flush` option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in `mysqld` that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as `mysqld` without locking the table properly.
- You are running many `mysqld` servers using the same data directory on a system that doesn't support good filesystem locks (normally handled by the `lockd` lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused `mysqld`.
- You have found a bug in the data storage code. This isn't likely, but it's at least possible. In this case, you can try to change the storage engine to another engine by using `ALTER TABLE` on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others crash for you. Please try the following things:

- Stop the `mysqld` server with `mysqladmin shutdown`, run `myisamchk --silent --force */*.MYI` from the data directory to check all `MyISAM` tables, and restart `mysqld`. This ensures that you are running from a clean state. See [Chapter 5, Database Administration](#).
- Start `mysqld` with the `--log` option and try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See [Section 5.11.3, “The General Query Log”](#). If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have been able to locate the bug and should submit a bug report for it. See [Section 1.8, “How to Report Bugs or Problems”](#).
- Try to make a test case that we can use to repeat the problem. See [MySQL Internals: Porting](#).
- Try running the tests in the `mysql-test` directory and the MySQL benchmarks. See [Section 28.1.2, “MySQL Test Suite”](#). They should test MySQL rather well. You can also add code to the benchmarks that simulates your application. The benchmarks can be found in the `sql-bench` directory in a source distribution or, for a binary distribution, in the `sql-bench` directory under your MySQL installation directory.
- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)
- If you configure MySQL for debugging, it is much easier to gather information about possible errors if something goes wrong. Configuring MySQL for debugging causes a safe memory allocator to be included that can find some errors. It also provides a lot of output about what is happening. Reconfigure MySQL with the `--with-debug` or `--with-debug=full` option to `configure` and then recompile. See [MySQL Internals: Porting](#).
- Make sure that you have applied the latest patches for your operating system.
- Use the `--skip-external-locking` option to `mysqld`. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` option tells `mysqld` not to use external locking. (This means that you cannot run two `mysqld` servers on the same data directory and that you must be careful if you use `myisamchk`. Nevertheless, it may be instructive to try the option as a test.)
- Have you tried `mysqladmin -u root processlist` when `mysqld` appears to be running but not responding? Sometimes `mysqld` is not comatose even though you might think so. The problem may be that all connections are in use, or there may be some internal lock problem. `mysqladmin -u root processlist` usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.
- Run the command `mysqladmin -i 5 status` or `mysqladmin -i 5 -r status` in a separate window to produce statistics while you run your other queries.

- Try the following:

1. Start `mysqld` from `gdb` (or another debugger). See [MySQL Internals: Porting](#).
2. Run your test scripts.
3. Print the backtrace and the local variables at the three lowest levels. In `gdb`, you can do this with the following commands when `mysqld` has crashed inside `gdb`:

```
backtrace
info local
up
info local
up
info local
```

With `gdb`, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread N`, where `N` is the thread ID.

- Try to simulate your application with a Perl script to force MySQL to crash or misbehave.
- Send a normal bug report. See [Section 1.8, “How to Report Bugs or Problems”](#). Be even more detailed than usual. Because MySQL works for many people, it may be that the crash results from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only `VARCHAR` columns (not `BLOB` or `TEXT` columns), you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

The current dynamic row code has been in use at MySQL AB for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.

- Do not rule out your server hardware when diagnosing problems. Defective hardware can be the cause of data corruption. Particular attention should be paid to both RAMS and hard-drives when troubleshooting hardware.

B.1.4.3. How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as “no space left on device”), and to quota-exceeded errors (such as “write failed” or “user block limit reached”).

This section is relevant for writes to `MyISAM` tables. It also applies for writes to binary log files and binary log index file, except that references to “row” and “record” should be understood to mean “event.”

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, you can take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- To abort the thread, you must use `mysqladmin kill`. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several “locked” threads, killing the one thread that is waiting on the disk-full condition allows the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA INFILE` or after an `ALTER TABLE` statement. All of these statements may create large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these

operations, it removes the big temporary files and mark the table as crashed. The exception is that for **ALTER TABLE**, the old table is left unchanged.

MySQL Enterprise

For early notification of possible problems with your MySQL configuration subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

B.1.4.4. Where MySQL Stores Temporary Files

MySQL uses the value of the **TMPDIR** environment variable as the pathname of the directory in which to store temporary files. If you don't have **TMPDIR** set, MySQL uses the system default, which is normally **/tmp**, **/var/tmp**, or **/usr/tmp**. If the filesystem containing your temporary file directory is too small, you can use the **--tmpdir** option to **mysqld** to specify a directory in a filesystem where you have enough space.

In MySQL 5.1, the **--tmpdir** option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (':') on Unix and semicolon characters(';') on Windows, NetWare, and OS/2. **Note:** To spread the load effectively, these paths should be located on different *physical* disks, not different partitions of the same disk.

If the MySQL server is acting as a replication slave, you should not set **--tmpdir** to point to a directory on a memory-based filesystem or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or **LOAD DATA INFILE** operations. If files in the temporary file directory are lost when the server restarts, replication fails.

MySQL creates all temporary files as hidden files. This ensures that the temporary files are removed if **mysqld** is terminated. The disadvantage of using hidden files is that you do not see a big temporary file that fills up the filesystem in which the temporary file directory is located.

MySQL Enterprise

Advisors provided by the MySQL Enterprise Monitor automatically detect excessive temporary table storage to disk. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

When sorting (**ORDER BY** or **GROUP BY**), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some **SELECT** queries, MySQL also creates temporary SQL tables. These are not hidden and have names of the form **SQL_***.

ALTER TABLE creates a temporary table in the same directory as the original table.

B.1.4.5. How to Protect or Change the MySQL Unix Socket File

The default location for the Unix socket file that the server uses for communication with local clients is **/tmp/mysql.sock**. (For some distribution formats, the directory might be different, such as **/var/lib/mysql** for RPMs.)

On some versions of Unix, anyone can delete files in the **/tmp** directory or other similar directories used for temporary files. If the socket file is located in such a directory on your system, this might cause problems.

On most versions of Unix, you can protect your **/tmp** directory so that files can be deleted only by their owners or the superuser (**root**). To do this, set the **sticky** bit on the **/tmp** directory by logging in as **root** and using the following command:

```
shell> chmod +t /tmp
```

You can check whether the **sticky** bit is set by executing **ls -ld /tmp**. If the last permission character is **t**, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in **/etc/my.cnf**:

```
[mysqld]
```

```
socket=/path/to/socket
[client]
socket=/path/to/socket
```

See [Section 4.3.2, “Using Option Files”](#).

- Specify a `--socket` option on the command line to `mysqld_safe` and when you run client programs.
- Set the `MYSQL_UNIX_PORT` environment variable to the path of the Unix socket file.
- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the `--with-unix-socket-path` option when you run `configure`. See [Section 2.9.2, “Typical configure Options”](#).

You can test whether the new socket location works by attempting to connect to the server with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

B.1.4.6. Time Zone Problems

If you have a problem with `SELECT NOW()` returning values in UTC and not your local time, you have to tell the server your current time zone. The same applies if `UNIX_TIMESTAMP()` returns the wrong value. This should be done for the environment in which the server runs; for example, in `mysqld_safe` or `mysql.server`. See [Section 2.14, “Environment Variables”](#).

You can set the time zone for the server with the `--timezone=timezone_name` option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`.

The allowable values for `--timezone` or `TZ` are system-dependent. Consult your operating system documentation to see what values are acceptable.

B.1.5. Query-Related Issues

B.1.5.1. Case Sensitivity in Searches

By default, MySQL searches are not case sensitive (although there are some character sets that are never case insensitive, such as `czech`). This means that if you search with `col_name LIKE 'a%'`, you get all column values that start with `A` or `a`. If you want to make this search case sensitive, make sure that one of the operands has a case sensitive or binary collation. For example, if you are comparing a column and a string that both have the `latin1` character set, you can use the `COLLATE` operator to cause either operand to have the `latin1_general_cs` or `latin1_bin` collation. For example:

```
col_name COLLATE latin1_general_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_general_cs
col_name COLLATE latin1_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case sensitive or binary collation. See [Section 12.1.10, “CREATE TABLE Syntax”](#).

Simple comparison operations (`>=`, `>`, `=`, `<`, `<=`, sorting, and grouping) are based on each character's “sort value.” Characters with the same sort value (such as `‘E’`, `‘e’`, and `‘Ä#Â@’`) are treated as the same character.

B.1.5.2. Problems Using DATE Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is allowed. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

```
mysql> SELECT * FROM tbl_name WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in a numeric context (and vice versa). It is also smart enough to allow a “relaxed” string form when updating and in a `WHERE` clause that compares a date to a `TIMESTAMP`, `DATE`, or `DATETIME` column. (“Relaxed form” means that any punctuation character may be used as the separator between parts. For example, `'2004-08-15'` and `'2004#08#15'` are equivalent.) MySQL can also convert a string containing no separators (such as

'20040815'), provided it makes sense as a date.

When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` to a constant string with the `<`, `<=`, `=`, `>=`, `>`, or `BETWEEN` operators, MySQL normally converts the string to an internal long integer for faster comparison (and also for a bit more “relaxed” string checking). However, this conversion is subject to the following exceptions:

- When you compare two columns
- When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` column to an expression
- When you use any other comparison method than those just listed, such as `IN` or `STRCMP()`.

For these exceptional cases, the comparison is done by converting the objects to strings and performing a string comparison.

To keep things safe, assume that strings are compared as strings and use the appropriate string functions if you want to compare a temporal value to a string.

The special date '0000-00-00' can be stored and retrieved as '0000-00-00'. When using a '0000-00-00' date through MySQL ODBC, it is automatically converted to `NULL` in MySQL ODBC 2.50.12 and above, because ODBC can't handle this kind of date.

Because MySQL performs the conversions described above, the following statements work:

```
mysql> INSERT INTO tbl_name (idate) VALUES (19970505);
mysql> INSERT INTO tbl_name (idate) VALUES ('19970505');
mysql> INSERT INTO tbl_name (idate) VALUES ('97-05-05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997.05.05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997 05 05');
mysql> INSERT INTO tbl_name (idate) VALUES ('0000-00-00');

mysql> SELECT idate FROM tbl_name WHERE idate >= '1997-05-05';
mysql> SELECT idate FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT MOD(idate,100) FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT idate FROM tbl_name WHERE idate >= '19970505';
```

However, the following does not work:

```
mysql> SELECT idate FROM tbl_name WHERE STRCMP(idate,'20030505')=0;
```

`STRCMP()` is a string function, so it converts `idate` to a string in 'YYYY-MM-DD' format and performs a string comparison. It does not convert '20030505' to the date '2003-05-05' and perform a date comparison.

If you are using the `ALLOW_INVALID_DATES` SQL mode, MySQL allows you to store dates that are given only limited checking: MySQL requires only that the day is in the range from 1 to 31 and the month is in the range from 1 to 12.

This makes MySQL very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation).

If you are not using the `NO_ZERO_IN_DATE` SQL mode, the day or month part can be zero. This is convenient if you want to store a birthdate in a `DATE` column and you know only part of the date.

If you are not using the `NO_ZERO_DATE` SQL mode, MySQL also allows you to store '0000-00-00' as a “dummy date.” This is in some cases more convenient than using `NULL` values.

If the date cannot be converted to any reasonable value, a 0 is stored in the `DATE` column, which is retrieved as '0000-00-00'. This is both a speed and a convenience issue. We believe that the database server's responsibility is to retrieve the same date you stored (even if the data was not logically correct in all cases). We think it is up to the application and not the server to check the dates.

If you want MySQL to check all dates and accept only legal dates (unless overridden by `IGNORE`), you should set `sql_mode` to "NO_ZERO_IN_DATE, NO_ZERO_DATE".

B.1.5.3. Problems with `NULL` Values

The concept of the `NULL` value is a common source of confusion for newcomers to SQL, who often think that `NULL` is the same thing as an empty string ''. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second inserts an empty string. The meaning of the first can be regarded as “phone number is not known” and the meaning of the second can be regarded as “the person is known to have no phone, and thus no phone number.”

To help with `NULL` handling, you can use the `IS NULL` and `IS NOT NULL` operators and the `IFNULL()` function.

In SQL, the `NULL` value is never true in comparison to any other value, even `NULL`. An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return `NULL`:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

If you want to search for column values that are `NULL`, you cannot use an `expr = NULL` test. The following statement returns no rows, because `expr = NULL` is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for `NULL` values, you must use the `IS NULL` test. The following statements show how to find the `NULL` phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = '';
```

See [Section 3.3.4.6, “Working with NULL Values”](#), for additional information and examples.

You can add an index on a column that can have `NULL` values if you are using the `MyISAM`, `InnoDB`, or `MEMORY` storage engine. Otherwise, you must declare an indexed column `NOT NULL`, and you cannot insert `NULL` into the column.

When reading data with `LOAD DATA INFILE`, empty or missing columns are updated with `' '`. If you want a `NULL` value in a column, you should use `\N` in the data file. The literal word “`NULL`” may also be used under some circumstances. See [Section 12.2.5, “LOAD DATA INFILE Syntax”](#).

When using `DISTINCT`, `GROUP BY`, or `ORDER BY`, all `NULL` values are regarded as equal.

When using `ORDER BY`, `NULL` values are presented first, or last if you specify `DESC` to sort in descending order.

Aggregate (summary) functions such as `COUNT()`, `MIN()`, and `SUM()` ignore `NULL` values. The exception to this is `COUNT(*)`, which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-`NULL` values in the `age` column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some data types, MySQL handles `NULL` values specially. If you insert `NULL` into a `TIMESTAMP` column, the current date and time is inserted. If you insert `NULL` into an integer column that has the `AUTO_INCREMENT` attribute, the next number in the sequence is inserted.

B.1.5.4. Problems with Column Aliases

You can use an alias to refer to a column in `GROUP BY`, `ORDER BY`, or `HAVING` clauses. Aliases can also be used to give columns better names:

```
SELECT SQRT(a*b) AS root FROM tbl_name GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL doesn't allow you to refer to a column alias in a `WHERE` clause. This restriction is imposed because when the `WHERE` code is executed, the column value may not yet be determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl_name WHERE cnt > 0 GROUP BY id;
```

The `WHERE` statement is executed to determine which rows should be included in the `GROUP BY` part, whereas `HAVING` is used to decide which rows from the result set should be used.

B.1.5.5. Rollback Failure for Non-Transactional Tables

If you receive the following message when trying to perform a `ROLLBACK`, it means that one or more of the tables you used in the transaction do not support transactions:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

These non-transactional tables are not affected by the `ROLLBACK` statement.

If you were not deliberately mixing transactional and non-transactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can happen if you try to create a table using a transactional storage engine that is not supported by your `mysqld` server (or that was disabled with a startup option). If `mysqld` doesn't support a storage engine, it instead creates the table as a `MyISAM` table, which is non-transactional.

You can check the storage engine for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

See [Section 12.5.4.28, “SHOW TABLE STATUS Syntax”](#), and [Section 12.5.4.9, “SHOW CREATE TABLE Syntax”](#).

You can check which storage engines your `mysqld` server supports by using this statement:

```
SHOW ENGINES;
```

You can also use the following statement, and check the value of the variable that is associated with the storage engine in which you are interested:

```
SHOW VARIABLES LIKE 'have_%';
```

For example, to determine whether the `InnoDB` storage engine is available, check the value of the `have_innodb` variable.

See [Section 12.5.4.14, “SHOW ENGINES Syntax”](#), and [Section 12.5.4.31, “SHOW VARIABLES Syntax”](#).

MySQL Enterprise

Ensure that your data is adequately protected by subscribing to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

B.1.5.6. Deleting Rows from Related Tables

If the total length of the `DELETE` statement for `related_table` is more than 1MB (the default value of the `max_allowed_packet` system variable), you should split it into smaller parts and execute multiple `DELETE` statements. You probably get the fastest `DELETE` by specifying only 100 to 1,000 `related_column` values per statement if the `related_column` is indexed. If the `related_column` isn't indexed, the speed is independent of the number of arguments in the `IN` clause.

B.1.5.7. Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that doesn't return any rows, you should use the following procedure to find out what is wrong:

1. Test the query with `EXPLAIN` to check whether you can find something that is obviously wrong. See [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#).
2. Select only those columns that are used in the `WHERE` clause.
3. Remove one table at a time from the query until it returns some rows. If the tables are large, it's a good idea to use `LIMIT 10` with the query.
4. Issue a `SELECT` for the column that should have matched a row against the table that was last removed from the query.
5. If you are comparing `FLOAT` or `DOUBLE` columns with numbers that have decimals, you can't use equality (`=`) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some

cases, changing the `FLOAT` to a `DOUBLE` fixes this. See [Section B.1.5.8, “Problems with Floating-Point Comparisons”](#).

- If you still can't figure out what's wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file by dumping the tables with `mysqldump --quick db_name tbl_name_1 ... tbl_name_n > query.sql`. Open the file in an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your `SELECT` statement at the end of the file.

Verify that the test file demonstrates the problem by executing these commands:

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

Attach the test file to a bug report, which you can file using the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

B.1.5.8. Problems with Floating-Point Comparisons

Floating-point numbers sometimes cause confusion because they are approximate. That is, they are not stored as exact values inside computer architecture. What you can see on the screen usually is not the exact value of the number. The `FLOAT` and `DOUBLE` data types are such. For `DECIMAL` columns, MySQL performs operations with a precision of 64 decimal digits, which should solve most common inaccuracy problems.

The following example demonstrates the problem using `DOUBLE`. It shows that are calculations that are done using floating-point operations are subject to floating-point error.

```
mysql> CREATE TABLE t1 (i INT, d1 DOUBLE, d2 DOUBLE);
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
-> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
-> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
-> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
-> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
-> (6, 0.00, 0.00), (6, -51.40, 0.00);

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
-> FROM t1 GROUP BY i HAVING a <> b;
```

i	a	b
1	21.4	21.4
2	76.8	76.8
3	7.4	7.4
4	15.4	15.4
5	7.2	7.2
6	-51.4	0

The result is correct. Although the first five records look like they should not satisfy the comparison (the values of `a` and `b` do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on factors such as computer architecture or the compiler version or optimization level. For example, different CPUs may evaluate floating-point numbers differently.

If columns `d1` and `d2` had been defined as `DECIMAL` rather than `DOUBLE`, the result of the `SELECT` query would have contained only one row — the last one shown above.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) > 0.0001;
```

i	a	b
6	-51.4	0

1 row in set (0.00 sec)

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
```

```
-> GROUP BY i HAVING ABS(a - b) <= 0.0001;
```

i	a	b
1	21.4	21.4
2	76.8	76.8
3	7.4	7.4
4	15.4	15.4
5	7.2	7.2

5 rows in set (0.03 sec)

B.1.6. Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL doesn't have enough information about the data at hand and has to make “educated” guesses about the data.

For the cases when MySQL does not do the “right” thing, tools that you have available to help MySQL are:

- Use the **EXPLAIN** statement to get information about how MySQL processes a query. To use it, just add the keyword **EXPLAIN** to the front of your **SELECT** statement:

```
mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
```

EXPLAIN is discussed in more detail in [Section 6.2.1, “Optimizing Queries with EXPLAIN”](#).

- Use **ANALYZE TABLE *tbl_name*** to update the key distributions for the scanned table. See [Section 12.5.2.1, “ANALYZE TABLE Syntax”](#).
- Use **FORCE INDEX** for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

USE INDEX and **IGNORE INDEX** may also be useful. See [Section 12.2.7.2, “Index Hint Syntax”](#).

- Global and table-level **STRAIGHT_JOIN**. See [Section 12.2.7, “SELECT Syntax”](#).
- You can tune global or thread-specific system variables. For example, Start **mysqld** with the **--max-seeks-for-key=1000** option or use **SET max_seeks_for_key=1000** to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See [Section 5.2.3, “System Variables”](#).

MySQL Enterprise

For expert advice on configuring MySQL servers for optimal performance, subscribe to the MySQL Enterprise Monitor. For more information see <http://www.mysql.com/products/enterprise/advisors.html>.

B.1.7. Table Definition-Related Issues

B.1.7.1. Problems with **ALTER TABLE**

ALTER TABLE changes a table to the current character set. If you get a duplicate-key error during **ALTER TABLE**, the cause is either that the new character sets maps two keys to the same value or that the table is corrupted. In the latter case, you should run **REPAIR TABLE** on the table.

If **ALTER TABLE** dies with the following error, the problem may be that MySQL crashed during an earlier **ALTER TABLE** operation and there is an old table named **A-xxx** or **B-xxx** lying around:

```
Error on rename of './database/name.frm'
to './database/B-xxx.frm' (Errcode: 17)
```

In this case, go to the MySQL data directory and delete all files that have names starting with **A-** or **B-**. (You may want to move them elsewhere instead of deleting them.)

ALTER TABLE works in the following way:

- Create a new table named `A-xxx` with the requested structural changes.
- Copy all rows from the original table to `A-xxx`.
- Rename the original table to `B-xxx`.
- Rename `A-xxx` to your original table name.
- Delete `B-xxx`.

If something goes wrong with the renaming operation, MySQL tries to undo the changes. If something goes seriously wrong (although this shouldn't happen), MySQL may leave the old table as `B-xxx`. A simple rename of the table files at the system level should get your data back.

If you use `ALTER TABLE` on a transactional table or if you are using Windows or OS/2, `ALTER TABLE` unlocks the table if you had done a `LOCK TABLE` on it. This is done because `InnoDB` and these operating systems cannot drop a table that is in use.

B.1.7.2. How to Change the Order of Columns in a Table

First, consider whether you really need to change the column order in a table. The whole point of SQL is to abstract the application from the data storage format. You should always specify the order in which you wish to retrieve your data. The first of the following statements returns columns in the order `col_name1, col_name2, col_name3`, whereas the second returns them in the order `col_name1, col_name3, col_name2`:

```
mysql> SELECT col_name1, col_name2, col_name3 FROM tbl_name;
mysql> SELECT col_name1, col_name3, col_name2 FROM tbl_name;
```

If you decide to change the order of table columns anyway, you can do so as follows:

1. Create a new table with the columns in the new order.
2. Execute this statement:

```
mysql> INSERT INTO new_table
-> SELECT columns-in-new-order FROM old_table;
```

3. Drop or rename `old_table`.
4. Rename the new table to the original name:

```
mysql> ALTER TABLE new_table RENAME old_table;
```

`SELECT *` is quite suitable for testing queries. However, in an application, you should *never* rely on using `SELECT *` and retrieving the columns based on their position. The order and position in which columns are returned does not remain the same if you add, move, or delete columns. A simple change to your table structure could cause your application to fail.

B.1.7.3. TEMPORARY TABLE Problems

The following list indicates limitations on the use of `TEMPORARY` tables:

- A `TEMPORARY` table can only be of type `MEMORY`, `MyISAM`, `MERGE`, or `InnoDB`.

Temporary tables are not supported for MySQL Cluster.

- You cannot refer to a `TEMPORARY` table more than once in the same query. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- The `SHOW TABLES` statement does not list `TEMPORARY` tables.

- You cannot use `RENAME` to rename a `TEMPORARY` table. However, you can use `ALTER TABLE` instead:

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

- There are known issues in using temporary tables with replication. See [Section 15.3.1, “Replication Features and Issues”](#), for more information.

B.1.8. Known Issues in MySQL

This section is a list of the known issues in recent versions of MySQL.

For information about platform-specific issues, see the installation and porting instructions in [Section 2.13, “Operating System-Specific Notes”](#), and [MySQL Internals: Porting](#).

B.1.8.1. Open Issues in MySQL

The following problems are known and fixing them is a high priority:

- MySQL Cluster fails to recover from an out-of-disk failure when using disk data. ([Bug#17614](#))
- Subquery optimization for `IN` is not as effective as for `=`.
- Even if you use `lower_case_table_names=2` (which enables MySQL to remember the case used for databases and table names), MySQL does not remember the case used for database names for the function `DATABASE()` or within the various logs (on case-insensitive systems).
- Dropping a `FOREIGN KEY` constraint doesn't work in replication because the constraint may have another name on the slave.
- `REPLACE` (and `LOAD DATA` with the `REPLACE` option) does not trigger `ON DELETE CASCADE`.
- `DISTINCT` with `ORDER BY` doesn't work inside `GROUP_CONCAT()` if you don't use all and only those columns that are in the `DISTINCT` list.
- If one user has a long-running transaction and another user drops a table that is updated in the transaction, there is small chance that the binary log may contain the `DROP TABLE` command before the table is used in the transaction itself. We plan to fix this by having the `DROP TABLE` command wait until the table is not being used in any transaction.
- When inserting a big integer value (between 2^{63} and $2^{64}-1$) into a decimal or string column, it is inserted as a negative value because the number is evaluated in a signed integer context.
- `FLUSH TABLES WITH READ LOCK` does not block `COMMIT` if the server is running without binary logging, which may cause a problem (of consistency between tables) when doing a full backup.
- `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` may cause problems on tables for which you are using `INSERT DELAYED`.
- Performing `LOCK TABLE ...` and `FLUSH TABLES ...` doesn't guarantee that there isn't a half-finished transaction in progress on the table.
- Replication uses query-level logging: The master writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases.

It is possible for the data on the master and slave to become different if a query is designed in such a way that the data modification is non-deterministic (generally not a recommended practice, even outside of replication).

For example:

- `CREATE ... SELECT` or `INSERT ... SELECT` statements that insert zero or `NULL` values into an `AUTO_INCREMENT` column.
- `DELETE` if you are deleting rows from a table that has foreign keys with `ON DELETE CASCADE` properties.
- `REPLACE ... SELECT`, `INSERT IGNORE ... SELECT` if you have duplicate key values in the inserted data.

If and only if the preceding queries have no **ORDER BY** clause guaranteeing a deterministic order.

For example, for **INSERT ... SELECT** with no **ORDER BY**, the **SELECT** may return rows in a different order (which results in a row having different ranks, hence getting a different number in the **AUTO_INCREMENT** column), depending on the choices made by the optimizers on the master and slave.

A query is optimized differently on the master and slave only if:

- The table is stored using a different storage engine on the master than on the slave. (It is possible to use different storage engines on the master and slave. For example, you can use **InnoDB** on the master, but **MyISAM** on the slave if the slave has less available disk space.)
- MySQL buffer sizes (**key_buffer_size**, and so on) are different on the master and slave.
- The master and slave run different MySQL versions, and the optimizer code differs between these versions.

This problem may also affect database restoration using **mysqlbinlog|mysql**.

The easiest way to avoid this problem is to add an **ORDER BY** clause to the aforementioned non-deterministic queries to ensure that the rows are always stored or modified in the same order.

In future MySQL versions, we will automatically add an **ORDER BY** clause when needed.

The following issues are known and will be fixed in due time:

- Log filenames are based on the server hostname (if you don't specify a filename with the startup option). You have to use options such as **--log-bin=old_host_name-bin** if you change your hostname to something else. Another option is to rename the old files to reflect your hostname change (if these are binary logs, you need to edit the binary log index file and fix the binlog names there as well). See [Section 5.2.2, “Command Options”](#).
- **mysqlbinlog** does not delete temporary files left after a **LOAD DATA INFILE** command. See [Section 7.11, “mysqlbinlog — Utility for Processing Binary Log Files”](#).
- **RENAME** doesn't work with **TEMPORARY** tables or tables used in a **MERGE** table.
- Due to the way table format (**.frm**) files are stored, you cannot use character 255 (**CHAR(255)**) in table names, column names, or enumerations. This is scheduled to be fixed in version 5.1 when we implement new table definition format files.
- When using **SET CHARACTER SET**, you can't use translated characters in database, table, and column names.
- You can't use **'_'** or **'%'** with **ESCAPE** in **LIKE ... ESCAPE**.
- You cannot build the server in another directory when using MIT-pthreads. Because this requires changes to MIT-pthreads, we are not likely to fix this. See [Section 2.9.5, “MIT-pthreads Notes”](#).
- **BLOB** and **TEXT** values can't reliably be used in **GROUP BY**, **ORDER BY** or **DISTINCT**. Only the first **max_sort_length** bytes are used when comparing **BLOB** values in these cases. The default value of **max_sort_length** is 1024 and can be changed at server startup time or at runtime.
- Numeric calculations are done with **BIGINT** or **DOUBLE** (both are normally 64 bits long). Which precision you get depends on the function. The general rule is that bit functions are performed with **BIGINT** precision, **IF** and **ELT()** with **BIGINT** or **DOUBLE** precision, and the rest with **DOUBLE** precision. You should try to avoid using unsigned long long values if they resolve to be larger than 63 bits (9223372036854775807) for anything other than bit fields.
- You can have up to 255 **ENUM** and **SET** columns in one table.
- In **MIN()**, **MAX()**, and other aggregate functions, MySQL currently compares **ENUM** and **SET** columns by their string value rather than by the string's relative position in the set.
- **mysqld_safe** redirects all messages from **mysqld** to the **mysqld** log. One problem with this is that if you execute **mysqldadmin refresh** to close and reopen the log, **stdout** and **stderr** are still redirected to the old log. If you use **--log** extensively, you should edit **mysqld_safe** to log to **host_name.err** instead of **host_name.log** so that you can easily reclaim the space for the old log by deleting it and executing **mysqldadmin refresh**.

- In an `UPDATE` statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments `KEY` by 2, **not** 1:

```
mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following doesn't work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- The optimizer may handle `DISTINCT` differently when you are using “hidden” columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns don't participate in the `DISTINCT` comparison. We will probably change this in the future to never compare the hidden columns when executing `DISTINCT`.

An example of this is:

```
SELECT DISTINCT mp3id FROM band_downloads
WHERE userid = 9 ORDER BY id DESC;
```

and

```
SELECT DISTINCT band_downloads.mp3id
FROM band_downloads,band_mp3
WHERE band_downloads.userid = 9
AND band_mp3.id = band_downloads.mp3id
ORDER BY band_downloads.id DESC;
```

In the second case, using MySQL Server 3.23.x, you may get two identical rows in the result set (because the values in the hidden `id` column may differ).

Note that this happens only for queries where that do not have the `ORDER BY` columns in the result.

- If you execute a `PROCEDURE` on a query that returns an empty set, in some cases the `PROCEDURE` does not transform the columns.
- Creation of a table of type `MERGE` doesn't check whether the underlying tables are compatible types.
- If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table and then add a normal index on the `MERGE` table, the key order is different for the tables if there was an old, non-`UNIQUE` key in the table. This is because `ALTER TABLE` puts `UNIQUE` indexes before normal indexes to be able to detect duplicate keys as early as possible.

B.2. Server Error Codes and Messages

MySQL programs have access to several types of error information when the server returns an error. For example, the `mysql` client program displays errors using the following format:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

The message displayed contains three types of information:

- A numeric error value (**1146**). This number is MySQL-specific and is not portable to other database systems.
- A five-character SQLSTATE value (`'42S02'`). The values are specified by ANSI SQL and ODBC and are more standardized. Not all MySQL error numbers are mapped to SQLSTATE error codes. The value `'HY000'` (general error) is used for unmapped errors.
- A string that provides a textual description of the error.

Server error information comes from the following source files. For details about the way that error information is defined, see the MySQL Internals manual, available at <http://dev.mysql.com/doc/>.

- Error message information is listed in the [share/errmsg.txt](#) file. %d and %s represent numbers and strings, respectively, that are substituted into the Message values when they are displayed.
- The Error values listed in [share/errmsg.txt](#) are used to generate the definitions in the [include/mysql_error.h](#) and [include/mysql_ename.h](#) MySQL source files.
- The SQLSTATE values listed in [share/errmsg.txt](#) are used to generate the definitions in the [include/sql_state.h](#) MySQL source file.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: [1000](#) SQLSTATE: [HY000](#) ([ER_HASHCHK](#))
Message: hashchk
- Error: [1001](#) SQLSTATE: [HY000](#) ([ER_NISAMCHK](#))
Message: isamchk
- Error: [1002](#) SQLSTATE: [HY000](#) ([ER_NO](#))
Message: NO
- Error: [1003](#) SQLSTATE: [HY000](#) ([ER_YES](#))
Message: YES
- Error: [1004](#) SQLSTATE: [HY000](#) ([ER_CANT_CREATE_FILE](#))
Message: Can't create file '%s' (errno: %d)
- Error: [1005](#) SQLSTATE: [HY000](#) ([ER_CANT_CREATE_TABLE](#))
Message: Can't create table '%s' (errno: %d)
- Error: [1006](#) SQLSTATE: [HY000](#) ([ER_CANT_CREATE_DB](#))
Message: Can't create database '%s' (errno: %d)
- Error: [1007](#) SQLSTATE: [HY000](#) ([ER_DB_CREATE_EXISTS](#))
Message: Can't create database '%s'; database exists
- Error: [1008](#) SQLSTATE: [HY000](#) ([ER_DB_DROP_EXISTS](#))
Message: Can't drop database '%s'; database doesn't exist
- Error: [1009](#) SQLSTATE: [HY000](#) ([ER_DB_DROP_DELETE](#))
Message: Error dropping database (can't delete '%s', errno: %d)
- Error: [1010](#) SQLSTATE: [HY000](#) ([ER_DB_DROP_RMDIR](#))
Message: Error dropping database (can't rmdir '%s', errno: %d)
- Error: [1011](#) SQLSTATE: [HY000](#) ([ER_CANT_DELETE_FILE](#))
Message: Error on delete of '%s' (errno: %d)
- Error: [1012](#) SQLSTATE: [HY000](#) ([ER_CANT_FIND_SYSTEM_REC](#))
Message: Can't read record in system table
- Error: [1013](#) SQLSTATE: [HY000](#) ([ER_CANT_GET_STAT](#))

Message: Can't get status of '%s' (errno: %d)

- Error: [1014](#) SQLSTATE: [HY000](#) ([ER_CANT_GET_WD](#))

Message: Can't get working directory (errno: %d)

- Error: [1015](#) SQLSTATE: [HY000](#) ([ER_CANT_LOCK](#))

Message: Can't lock file (errno: %d)

- Error: [1016](#) SQLSTATE: [HY000](#) ([ER_CANT_OPEN_FILE](#))

Message: Can't open file: '%s' (errno: %d)

- Error: [1017](#) SQLSTATE: [HY000](#) ([ER_FILE_NOT_FOUND](#))

Message: Can't find file: '%s' (errno: %d)

- Error: [1018](#) SQLSTATE: [HY000](#) ([ER_CANT_READ_DIR](#))

Message: Can't read dir of '%s' (errno: %d)

- Error: [1019](#) SQLSTATE: [HY000](#) ([ER_CANT_SET_WD](#))

Message: Can't change dir to '%s' (errno: %d)

- Error: [1020](#) SQLSTATE: [HY000](#) ([ER_CHECKREAD](#))

Message: Record has changed since last read in table '%s'

- Error: [1021](#) SQLSTATE: [HY000](#) ([ER_DISK_FULL](#))

Message: Disk full (%s); waiting for someone to free some space...

- Error: [1022](#) SQLSTATE: [23000](#) ([ER_DUP_KEY](#))

Message: Can't write; duplicate key in table '%s'

- Error: [1023](#) SQLSTATE: [HY000](#) ([ER_ERROR_ON_CLOSE](#))

Message: Error on close of '%s' (errno: %d)

- Error: [1024](#) SQLSTATE: [HY000](#) ([ER_ERROR_ON_READ](#))

Message: Error reading file '%s' (errno: %d)

- Error: [1025](#) SQLSTATE: [HY000](#) ([ER_ERROR_ON_RENAME](#))

Message: Error on rename of '%s' to '%s' (errno: %d)

- Error: [1026](#) SQLSTATE: [HY000](#) ([ER_ERROR_ON_WRITE](#))

Message: Error writing file '%s' (errno: %d)

- Error: [1027](#) SQLSTATE: [HY000](#) ([ER_FILE_USED](#))

Message: '%s' is locked against change

- Error: [1028](#) SQLSTATE: [HY000](#) ([ER_FILSORT_ABORT](#))

Message: Sort aborted

- Error: [1029](#) SQLSTATE: [HY000](#) ([ER_FORM_NOT_FOUND](#))

Message: View '%s' doesn't exist for '%s'

- Error: **1030** SQLSTATE: **HY000** (**ER_GET_ERRNO**)
Message: Got error %d from storage engine
- Error: **1031** SQLSTATE: **HY000** (**ER_ILLEGAL HA**)
Message: Table storage engine for '%s' doesn't have this option
- Error: **1032** SQLSTATE: **HY000** (**ER_KEY_NOT_FOUND**)
Message: Can't find record in '%s'
- Error: **1033** SQLSTATE: **HY000** (**ER_NOT_FORM_FILE**)
Message: Incorrect information in file: '%s'
- Error: **1034** SQLSTATE: **HY000** (**ER_NOT_KEYFILE**)
Message: Incorrect key file for table '%s'; try to repair it
- Error: **1035** SQLSTATE: **HY000** (**ER_OLD_KEYFILE**)
Message: Old key file for table '%s'; repair it!
- Error: **1036** SQLSTATE: **HY000** (**ER_OPEN_AS_READONLY**)
Message: Table '%s' is read only
- Error: **1037** SQLSTATE: **HY001** (**ER_OUTOFMEMORY**)
Message: Out of memory; restart server and try again (needed %d bytes)
- Error: **1038** SQLSTATE: **HY001** (**ER_OUT_OF_SORTMEMORY**)
Message: Out of sort memory; increase server sort buffer size
- Error: **1039** SQLSTATE: **HY000** (**ER_UNEXPECTED_EOF**)
Message: Unexpected EOF found when reading file '%s' (errno: %d)
- Error: **1040** SQLSTATE: **08004** (**ER_CON_COUNT_ERROR**)
Message: Too many connections
- Error: **1041** SQLSTATE: **HY000** (**ER_OUT_OF_RESOURCES**)
Message: Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space
- Error: **1042** SQLSTATE: **08S01** (**ER_BAD_HOST_ERROR**)
Message: Can't get hostname for your address
- Error: **1043** SQLSTATE: **08S01** (**ER_HANDSHAKE_ERROR**)
Message: Bad handshake
- Error: **1044** SQLSTATE: **42000** (**ER_DBACCESS_DENIED_ERROR**)
Message: Access denied for user '%s'@'%s' to database '%s'
- Error: **1045** SQLSTATE: **28000** (**ER_ACCESS_DENIED_ERROR**)
Message: Access denied for user '%s'@'%s' (using password: %s)
- Error: **1046** SQLSTATE: **3D000** (**ER_NO_DB_ERROR**)

Message: No database selected

- Error: [1047](#) SQLSTATE: [08S01](#) ([ER_UNKNOWN_COM_ERROR](#))

Message: Unknown command

- Error: [1048](#) SQLSTATE: [23000](#) ([ER_BAD_NULL_ERROR](#))

Message: Column '%s' cannot be null

- Error: [1049](#) SQLSTATE: [42000](#) ([ER_BAD_DB_ERROR](#))

Message: Unknown database '%s'

- Error: [1050](#) SQLSTATE: [42S01](#) ([ER_TABLE_EXISTS_ERROR](#))

Message: Table '%s' already exists

- Error: [1051](#) SQLSTATE: [42S02](#) ([ER_BAD_TABLE_ERROR](#))

Message: Unknown table '%s'

- Error: [1052](#) SQLSTATE: [23000](#) ([ER_NON_UNIQ_ERROR](#))

Message: Column '%s' in %s is ambiguous

- Error: [1053](#) SQLSTATE: [08S01](#) ([ER_SERVER_SHUTDOWN](#))

Message: Server shutdown in progress

- Error: [1054](#) SQLSTATE: [42S22](#) ([ER_BAD_FIELD_ERROR](#))

Message: Unknown column '%s' in '%s'

- Error: [1055](#) SQLSTATE: [42000](#) ([ER_WRONG_FIELD_WITH_GROUP](#))

Message: '%s' isn't in GROUP BY

- Error: [1056](#) SQLSTATE: [42000](#) ([ER_WRONG_GROUP_FIELD](#))

Message: Can't group on '%s'

- Error: [1057](#) SQLSTATE: [42000](#) ([ER_WRONG_SUM_SELECT](#))

Message: Statement has sum functions and columns in same statement

- Error: [1058](#) SQLSTATE: [21S01](#) ([ER_WRONG_VALUE_COUNT](#))

Message: Column count doesn't match value count

- Error: [1059](#) SQLSTATE: [42000](#) ([ER_TOO_LONG_IDENT](#))

Message: Identifier name '%s' is too long

- Error: [1060](#) SQLSTATE: [42S21](#) ([ER_DUP_FIELDNAME](#))

Message: Duplicate column name '%s'

- Error: [1061](#) SQLSTATE: [42000](#) ([ER_DUP_KEYNAME](#))

Message: Duplicate key name '%s'

- Error: [1062](#) SQLSTATE: [23000](#) ([ER_DUP_ENTRY](#))

Message: Duplicate entry '%s' for key %d

- Error: **1063** SQLSTATE: **42000** (**ER_WRONG_FIELD_SPEC**)
Message: Incorrect column specifier for column '%s'
- Error: **1064** SQLSTATE: **42000** (**ER_PARSE_ERROR**)
Message: %s near '%s' at line %d
- Error: **1065** SQLSTATE: **42000** (**ER_EMPTY_QUERY**)
Message: Query was empty
- Error: **1066** SQLSTATE: **42000** (**ER_NONUNIQ_TABLE**)
Message: Not unique table/alias: '%s'
- Error: **1067** SQLSTATE: **42000** (**ER_INVALID_DEFAULT**)
Message: Invalid default value for '%s'
- Error: **1068** SQLSTATE: **42000** (**ER_MULTIPLE_PRI_KEY**)
Message: Multiple primary key defined
- Error: **1069** SQLSTATE: **42000** (**ER_TOO_MANY_KEYS**)
Message: Too many keys specified; max %d keys allowed
- Error: **1070** SQLSTATE: **42000** (**ER_TOO_MANY_KEY_PARTS**)
Message: Too many key parts specified; max %d parts allowed
- Error: **1071** SQLSTATE: **42000** (**ER_TOO_LONG_KEY**)
Message: Specified key was too long; max key length is %d bytes
- Error: **1072** SQLSTATE: **42000** (**ER_KEY_COLUMN_DOES_NOT_EXISTS**)
Message: Key column '%s' doesn't exist in table
- Error: **1073** SQLSTATE: **42000** (**ER_BLOB_USED_AS_KEY**)
Message: BLOB column '%s' can't be used in key specification with the used table type
- Error: **1074** SQLSTATE: **42000** (**ER_TOO_BIG_FIELDLENGTH**)
Message: Column length too big for column '%s' (max = %d); use BLOB or TEXT instead
- Error: **1075** SQLSTATE: **42000** (**ER_WRONG_AUTO_KEY**)
Message: Incorrect table definition; there can be only one auto column and it must be defined as a key
- Error: **1076** SQLSTATE: **HY000** (**ER_READY**)
Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d
- Error: **1077** SQLSTATE: **HY000** (**ER_NORMAL_SHUTDOWN**)
Message: %s: Normal shutdown
- Error: **1078** SQLSTATE: **HY000** (**ER_GOT_SIGNAL**)
Message: %s: Got signal %d. Aborting!
- Error: **1079** SQLSTATE: **HY000** (**ER_SHUTDOWN_COMPLETE**)
Message: %s: Shutdown complete

- Error: **1080** SQLSTATE: **08S01** (**ER_FORCING_CLOSE**)
Message: %s: Forcing close of thread %ld user: '%s'
- Error: **1081** SQLSTATE: **08S01** (**ER_IPSOCK_ERROR**)
Message: Can't create IP socket
- Error: **1082** SQLSTATE: **42S12** (**ER_NO_SUCH_INDEX**)
Message: Table '%s' has no index like the one used in CREATE INDEX; recreate the table
- Error: **1083** SQLSTATE: **42000** (**ER_WRONG_FIELD_TERMINATORS**)
Message: Field separator argument is not what is expected; check the manual
- Error: **1084** SQLSTATE: **42000** (**ER_BLOBS_AND_NO_TERMINATED**)
Message: You can't use fixed rowlength with BLOBs; please use 'fields terminated by'
- Error: **1085** SQLSTATE: **HY000** (**ER_TEXTFILE_NOT_READABLE**)
Message: The file '%s' must be in the database directory or be readable by all
- Error: **1086** SQLSTATE: **HY000** (**ER_FILE_EXISTS_ERROR**)
Message: File '%s' already exists
- Error: **1087** SQLSTATE: **HY000** (**ER_LOAD_INFO**)
Message: Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld
- Error: **1088** SQLSTATE: **HY000** (**ER_ALTER_INFO**)
Message: Records: %ld Duplicates: %ld
- Error: **1089** SQLSTATE: **HY000** (**ER_WRONG_SUB_KEY**)
Message: Incorrect sub part key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique sub keys
- Error: **1090** SQLSTATE: **42000** (**ER_CANT_REMOVE_ALL_FIELDS**)
Message: You can't delete all columns with ALTER TABLE; use DROP TABLE instead
- Error: **1091** SQLSTATE: **42000** (**ER_CANT_DROP_FIELD_OR_KEY**)
Message: Can't DROP '%s'; check that column/key exists
- Error: **1092** SQLSTATE: **HY000** (**ER_INSERT_INFO**)
Message: Records: %ld Duplicates: %ld Warnings: %ld
- Error: **1093** SQLSTATE: **HY000** (**ER_UPDATE_TABLE_USED**)
Message: You can't specify target table '%s' for update in FROM clause
- Error: **1094** SQLSTATE: **HY000** (**ER_NO_SUCH_THREAD**)
Message: Unknown thread id: %lu
- Error: **1095** SQLSTATE: **HY000** (**ER_KILL_DENIED_ERROR**)
Message: You are not owner of thread %lu
- Error: **1096** SQLSTATE: **HY000** (**ER_NO_TABLES_USED**)

Message: No tables used

- Error: **1097** SQLSTATE: **HY000** (**ER_TOO_BIG_SET**)

Message: Too many strings for column %s and SET

- Error: **1098** SQLSTATE: **HY000** (**ER_NO_UNIQUE_LOGFILE**)

Message: Can't generate a unique log-filename %s.(1-999)

- Error: **1099** SQLSTATE: **HY000** (**ER_TABLE_NOT_LOCKED_FOR_WRITE**)

Message: Table '%s' was locked with a READ lock and can't be updated

- Error: **1100** SQLSTATE: **HY000** (**ER_TABLE_NOT_LOCKED**)

Message: Table '%s' was not locked with LOCK TABLES

- Error: **1101** SQLSTATE: **42000** (**ER_BLOB_CANT_HAVE_DEFAULT**)

Message: BLOB/TEXT column '%s' can't have a default value

- Error: **1102** SQLSTATE: **42000** (**ER_WRONG_DB_NAME**)

Message: Incorrect database name '%s'

- Error: **1103** SQLSTATE: **42000** (**ER_WRONG_TABLE_NAME**)

Message: Incorrect table name '%s'

- Error: **1104** SQLSTATE: **42000** (**ER_TOO_BIG_SELECT**)

Message: The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET SQL_MAX_JOIN_SIZE=# if the SELECT is okay

- Error: **1105** SQLSTATE: **HY000** (**ER_UNKNOWN_ERROR**)

Message: Unknown error

- Error: **1106** SQLSTATE: **42000** (**ER_UNKNOWN_PROCEDURE**)

Message: Unknown procedure '%s'

- Error: **1107** SQLSTATE: **42000** (**ER_WRONG_PARAMCOUNT_TO_PROCEDURE**)

Message: Incorrect parameter count to procedure '%s'

- Error: **1108** SQLSTATE: **HY000** (**ER_WRONG_PARAMETERS_TO_PROCEDURE**)

Message: Incorrect parameters to procedure '%s'

- Error: **1109** SQLSTATE: **42S02** (**ER_UNKNOWN_TABLE**)

Message: Unknown table '%s' in %s

- Error: **1110** SQLSTATE: **42000** (**ER_FIELD_SPECIFIED_TWICE**)

Message: Column '%s' specified twice

- Error: **1111** SQLSTATE: **HY000** (**ER_INVALID_GROUP_FUNC_USE**)

Message: Invalid use of group function

- Error: **1112** SQLSTATE: **42000** (**ER_UNSUPPORTED_EXTENSION**)

Message: Table '%s' uses an extension that doesn't exist in this MySQL version

- Error: [1113](#) SQLSTATE: [42000](#) ([ER_TABLE_MUST_HAVE_COLUMNS](#))
Message: A table must have at least 1 column
- Error: [1114](#) SQLSTATE: [HY000](#) ([ER_RECORD_FILE_FULL](#))
Message: The table '%s' is full
- Error: [1115](#) SQLSTATE: [42000](#) ([ER_UNKNOWN_CHARACTER_SET](#))
Message: Unknown character set: '%s'
- Error: [1116](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_TABLES](#))
Message: Too many tables; MySQL can only use %d tables in a join
- Error: [1117](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_FIELDS](#))
Message: Too many columns
- Error: [1118](#) SQLSTATE: [42000](#) ([ER_TOO_BIG_ROW_SIZE](#))
Message: Row size too large. The maximum row size for the used table type, not counting BLOBs, is %ld. You have to change some columns to TEXT or BLOBs
- Error: [1119](#) SQLSTATE: [HY000](#) ([ER_STACK_OVERRUN](#))
Message: Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld -O thread_stack=#' to specify a bigger stack if needed
- Error: [1120](#) SQLSTATE: [42000](#) ([ER_WRONG_OUTER_JOIN](#))
Message: Cross dependency found in OUTER JOIN; examine your ON conditions
- Error: [1121](#) SQLSTATE: [42000](#) ([ER_NULL_COLUMN_IN_INDEX](#))
Message: Table handler doesn't support NULL in given index. Please change column '%s' to be NOT NULL or use another handler
- Error: [1122](#) SQLSTATE: [HY000](#) ([ER_CANT_FIND_UDF](#))
Message: Can't load function '%s'
- Error: [1123](#) SQLSTATE: [HY000](#) ([ER_CANT_INITIALIZE_UDF](#))
Message: Can't initialize function '%s'; %s
- Error: [1124](#) SQLSTATE: [HY000](#) ([ER_UDF_NO_PATHS](#))
Message: No paths allowed for shared library
- Error: [1125](#) SQLSTATE: [HY000](#) ([ER_UDF_EXISTS](#))
Message: Function '%s' already exists
- Error: [1126](#) SQLSTATE: [HY000](#) ([ER_CANT_OPEN_LIBRARY](#))
Message: Can't open shared library '%s' (errno: %d %s)
- Error: [1127](#) SQLSTATE: [HY000](#) ([ER_CANT_FIND_DL_ENTRY](#))
Message: Can't find symbol '%s' in library
- Error: [1128](#) SQLSTATE: [HY000](#) ([ER_FUNCTION_NOT_DEFINED](#))
Message: Function '%s' is not defined
- Error: [1129](#) SQLSTATE: [HY000](#) ([ER_HOST_IS_BLOCKED](#))

Message: Host '%s' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'

- Error: **1130** SQLSTATE: **HY000** (**ER_HOST_NOT_PRIVILEGED**)

Message: Host '%s' is not allowed to connect to this MySQL server

- Error: **1131** SQLSTATE: **42000** (**ER_PASSWORD_ANONYMOUS_USER**)

Message: You are using MySQL as an anonymous user and anonymous users are not allowed to change passwords

- Error: **1132** SQLSTATE: **42000** (**ER_PASSWORD_NOT_ALLOWED**)

Message: You must have privileges to update tables in the mysql database to be able to change passwords for others

- Error: **1133** SQLSTATE: **42000** (**ER_PASSWORD_NO_MATCH**)

Message: Can't find any matching row in the user table

- Error: **1134** SQLSTATE: **HY000** (**ER_UPDATE_INFO**)

Message: Rows matched: %ld Changed: %ld Warnings: %ld

- Error: **1135** SQLSTATE: **HY000** (**ER_CANT_CREATE_THREAD**)

Message: Can't create a new thread (errno %d); if you are not out of available memory, you can consult the manual for a possible OS-dependent bug

- Error: **1136** SQLSTATE: **21S01** (**ER_WRONG_VALUE_COUNT_ON_ROW**)

Message: Column count doesn't match value count at row %ld

- Error: **1137** SQLSTATE: **HY000** (**ER_CANT_REOPEN_TABLE**)

Message: Can't reopen table: '%s'

- Error: **1138** SQLSTATE: **22004** (**ER_INVALID_USE_OF_NULL**)

Message: Invalid use of NULL value

- Error: **1139** SQLSTATE: **42000** (**ER_REGEXP_ERROR**)

Message: Got error '%s' from regexp

- Error: **1140** SQLSTATE: **42000** (**ER_MIX_OF_GROUP_FUNC_AND_FIELDS**)

Message: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause

- Error: **1141** SQLSTATE: **42000** (**ER_NONEXISTING_GRANT**)

Message: There is no such grant defined for user '%s' on host '%s'

- Error: **1142** SQLSTATE: **42000** (**ER_TABLEACCESS_DENIED_ERROR**)

Message: %s command denied to user '%s'@'%s' for table '%s'

- Error: **1143** SQLSTATE: **42000** (**ER_COLUMNACCESS_DENIED_ERROR**)

Message: %s command denied to user '%s'@'%s' for column '%s' in table '%s'

- Error: **1144** SQLSTATE: **42000** (**ER_ILLEGAL_GRANT_FOR_TABLE**)

Message: Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used

- Error: **1145** SQLSTATE: **42000** (**ER_GRANT_WRONG_HOST_OR_USER**)

Message: The host or user argument to GRANT is too long

- Error: [1146](#) SQLSTATE: [42S02](#) ([ER_NO_SUCH_TABLE](#))

Message: Table '%s.%s' doesn't exist

- Error: [1147](#) SQLSTATE: [42000](#) ([ER_NONEXISTING_TABLE_GRANT](#))

Message: There is no such grant defined for user '%s' on host '%s' on table '%s'

- Error: [1148](#) SQLSTATE: [42000](#) ([ER_NOT_ALLOWED_COMMAND](#))

Message: The used command is not allowed with this MySQL version

- Error: [1149](#) SQLSTATE: [42000](#) ([ER_SYNTAX_ERROR](#))

Message: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use

- Error: [1150](#) SQLSTATE: [HY000](#) ([ER_DELAYED_CANT_CHANGE_LOCK](#))

Message: Delayed insert thread couldn't get requested lock for table %s

- Error: [1151](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_DELAYED_THREADS](#))

Message: Too many delayed threads in use

- Error: [1152](#) SQLSTATE: [08S01](#) ([ER_ABORTING_CONNECTION](#))

Message: Aborted connection %ld to db: '%s' user: '%s' (%s)

- Error: [1153](#) SQLSTATE: [08S01](#) ([ER_NET_PACKET_TOO_LARGE](#))

Message: Got a packet bigger than 'max_allowed_packet' bytes

- Error: [1154](#) SQLSTATE: [08S01](#) ([ER_NET_READ_ERROR_FROM_PIPE](#))

Message: Got a read error from the connection pipe

- Error: [1155](#) SQLSTATE: [08S01](#) ([ER_NET_FCNTL_ERROR](#))

Message: Got an error from fcntl()

- Error: [1156](#) SQLSTATE: [08S01](#) ([ER_NET_PACKETS_OUT_OF_ORDER](#))

Message: Got packets out of order

- Error: [1157](#) SQLSTATE: [08S01](#) ([ER_NET_UNCOMPRESS_ERROR](#))

Message: Couldn't uncompress communication packet

- Error: [1158](#) SQLSTATE: [08S01](#) ([ER_NET_READ_ERROR](#))

Message: Got an error reading communication packets

- Error: [1159](#) SQLSTATE: [08S01](#) ([ER_NET_READ_INTERRUPTED](#))

Message: Got timeout reading communication packets

- Error: [1160](#) SQLSTATE: [08S01](#) ([ER_NET_ERROR_ON_WRITE](#))

Message: Got an error writing communication packets

- Error: [1161](#) SQLSTATE: [08S01](#) ([ER_NET_WRITE_INTERRUPTED](#))

Message: Got timeout writing communication packets

- Error: [1162](#) SQLSTATE: [42000](#) ([ER_TOO_LONG_STRING](#))
Message: Result string is longer than 'max_allowed_packet' bytes
- Error: [1163](#) SQLSTATE: [42000](#) ([ER_TABLE_CANT_HANDLE_BLOB](#))
Message: The used table type doesn't support BLOB/TEXT columns
- Error: [1164](#) SQLSTATE: [42000](#) ([ER_TABLE_CANT_HANDLE_AUTO_INCREMENT](#))
Message: The used table type doesn't support AUTO_INCREMENT columns
- Error: [1165](#) SQLSTATE: [HY000](#) ([ER_DELAYED_INSERT_TABLE_LOCKED](#))
Message: INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES
- Error: [1166](#) SQLSTATE: [42000](#) ([ER_WRONG_COLUMN_NAME](#))
Message: Incorrect column name '%s'
- Error: [1167](#) SQLSTATE: [42000](#) ([ER_WRONG_KEY_COLUMN](#))
Message: The used storage engine can't index column '%s'
- Error: [1168](#) SQLSTATE: [HY000](#) ([ER_WRONG_MRG_TABLE](#))
Message: Unable to open underlying table which is differently defined or of non-MyISAM type or doesn't exist
- Error: [1169](#) SQLSTATE: [23000](#) ([ER_DUP_UNIQUE](#))
Message: Can't write, because of unique constraint, to table '%s'
- Error: [1170](#) SQLSTATE: [42000](#) ([ER_BLOB_KEY_WITHOUT_LENGTH](#))
Message: BLOB/TEXT column '%s' used in key specification without a key length
- Error: [1171](#) SQLSTATE: [42000](#) ([ER_PRIMARY_CANT_HAVE_NULL](#))
Message: All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead
- Error: [1172](#) SQLSTATE: [42000](#) ([ER_TOO_MANY_ROWS](#))
Message: Result consisted of more than one row
- Error: [1173](#) SQLSTATE: [42000](#) ([ER_REQUIRES_PRIMARY_KEY](#))
Message: This table type requires a primary key
- Error: [1174](#) SQLSTATE: [HY000](#) ([ER_NO_RAID_COMPILED](#))
Message: This version of MySQL is not compiled with RAID support
- Error: [1175](#) SQLSTATE: [HY000](#) ([ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE](#))
Message: You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column
- Error: [1176](#) SQLSTATE: [42000](#) ([ER_KEY_DOES_NOT_EXISTS](#))
Message: Key '%s' doesn't exist in table '%s'
- Error: [1177](#) SQLSTATE: [42000](#) ([ER_CHECK_NO_SUCH_TABLE](#))
Message: Can't open table
- Error: [1178](#) SQLSTATE: [42000](#) ([ER_CHECK_NOT_IMPLEMENTED](#))
Message: The storage engine for the table doesn't support %s

- Error: **1179** SQLSTATE: **25000** (**ER_CANT_DO_THIS_DURING_AN_TRANSACTION**)
Message: You are not allowed to execute this command in a transaction
- Error: **1180** SQLSTATE: **HY000** (**ER_ERROR_DURING_COMMIT**)
Message: Got error %d during COMMIT
- Error: **1181** SQLSTATE: **HY000** (**ER_ERROR_DURING_ROLLBACK**)
Message: Got error %d during ROLLBACK
- Error: **1182** SQLSTATE: **HY000** (**ER_ERROR_DURING_FLUSH_LOGS**)
Message: Got error %d during FLUSH_LOGS
- Error: **1183** SQLSTATE: **HY000** (**ER_ERROR_DURING_CHECKPOINT**)
Message: Got error %d during CHECKPOINT
- Error: **1184** SQLSTATE: **08S01** (**ER_NEW_ABORTING_CONNECTION**)
Message: Aborted connection %ld to db: '%s' user: '%s' host: '%s' (%s)
- Error: **1185** SQLSTATE: **HY000** (**ER_DUMP_NOT_IMPLEMENTED**)
Message: The storage engine for the table does not support binary table dump
- Error: **1186** SQLSTATE: **HY000** (**ER_FLUSH_MASTER_BINLOG_CLOSED**)
Message: Binlog closed, cannot RESET MASTER
- Error: **1187** SQLSTATE: **HY000** (**ER_INDEX_REBUILD**)
Message: Failed rebuilding the index of dumped table '%s'
- Error: **1188** SQLSTATE: **HY000** (**ER_MASTER**)
Message: Error from master: '%s'
- Error: **1189** SQLSTATE: **08S01** (**ER_MASTER_NET_READ**)
Message: Net error reading from master
- Error: **1190** SQLSTATE: **08S01** (**ER_MASTER_NET_WRITE**)
Message: Net error writing to master
- Error: **1191** SQLSTATE: **HY000** (**ER_FT_MATCHING_KEY_NOT_FOUND**)
Message: Can't find FULLTEXT index matching the column list
- Error: **1192** SQLSTATE: **HY000** (**ER_LOCK_OR_ACTIVE_TRANSACTION**)
Message: Can't execute the given command because you have active locked tables or an active transaction
- Error: **1193** SQLSTATE: **HY000** (**ER_UNKNOWN_SYSTEM_VARIABLE**)
Message: Unknown system variable '%s'
- Error: **1194** SQLSTATE: **HY000** (**ER_CRASHED_ON_USAGE**)
Message: Table '%s' is marked as crashed and should be repaired
- Error: **1195** SQLSTATE: **HY000** (**ER_CRASHED_ON_REPAIR**)
Message: Table '%s' is marked as crashed and last (automatic?) repair failed

- Error: [1196](#) SQLSTATE: [HY000](#) ([ER_WARNING_NOT_COMPLETE_ROLLBACK](#))
Message: Some non-transactional changed tables couldn't be rolled back
- Error: [1197](#) SQLSTATE: [HY000](#) ([ER_TRANS_CACHE_FULL](#))
Message: Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again
- Error: [1198](#) SQLSTATE: [HY000](#) ([ER_SLAVE_MUST_STOP](#))
Message: This operation cannot be performed with a running slave; run STOP SLAVE first
- Error: [1199](#) SQLSTATE: [HY000](#) ([ER_SLAVE_NOT_RUNNING](#))
Message: This operation requires a running slave; configure slave and do START SLAVE
- Error: [1200](#) SQLSTATE: [HY000](#) ([ER_BAD_SLAVE](#))
Message: The server is not configured as slave; fix in config file or with CHANGE MASTER TO
- Error: [1201](#) SQLSTATE: [HY000](#) ([ER_MASTER_INFO](#))
Message: Could not initialize master info structure; more error messages can be found in the MySQL error log
- Error: [1202](#) SQLSTATE: [HY000](#) ([ER_SLAVE_THREAD](#))
Message: Could not create slave thread; check system resources
- Error: [1203](#) SQLSTATE: [42000](#) ([ER_TOO_MANY_USER_CONNECTIONS](#))
Message: User %s already has more than 'max_user_connections' active connections
- Error: [1204](#) SQLSTATE: [HY000](#) ([ER_SET_CONSTANTS_ONLY](#))
Message: You may only use constant expressions with SET
- Error: [1205](#) SQLSTATE: [HY000](#) ([ER_LOCK_WAIT_TIMEOUT](#))
Message: Lock wait timeout exceeded; try restarting transaction
- Error: [1206](#) SQLSTATE: [HY000](#) ([ER_LOCK_TABLE_FULL](#))
Message: The total number of locks exceeds the lock table size
- Error: [1207](#) SQLSTATE: [25000](#) ([ER_READ_ONLY_TRANSACTION](#))
Message: Update locks cannot be acquired during a READ UNCOMMITTED transaction
- Error: [1208](#) SQLSTATE: [HY000](#) ([ER_DROP_DB_WITH_READ_LOCK](#))
Message: DROP DATABASE not allowed while thread is holding global read lock
- Error: [1209](#) SQLSTATE: [HY000](#) ([ER_CREATE_DB_WITH_READ_LOCK](#))
Message: CREATE DATABASE not allowed while thread is holding global read lock
- Error: [1210](#) SQLSTATE: [HY000](#) ([ER_WRONG_ARGUMENTS](#))
Message: Incorrect arguments to %s
- Error: [1211](#) SQLSTATE: [42000](#) ([ER_NO_PERMISSION_TO_CREATE_USER](#))
Message: '%s'@'%s' is not allowed to create new users
- Error: [1212](#) SQLSTATE: [HY000](#) ([ER_UNION_TABLES_IN_DIFFERENT_DIR](#))

Message: Incorrect table definition; all MERGE tables must be in the same database

- Error: [1213](#) SQLSTATE: [40001](#) ([ER_LOCK_DEADLOCK](#))

Message: Deadlock found when trying to get lock; try restarting transaction

- Error: [1214](#) SQLSTATE: [HY000](#) ([ER_TABLE_CANT_HANDLE_FT](#))

Message: The used table type doesn't support FULLTEXT indexes

- Error: [1215](#) SQLSTATE: [HY000](#) ([ER_CANNOT_ADD_FOREIGN](#))

Message: Cannot add foreign key constraint

- Error: [1216](#) SQLSTATE: [23000](#) ([ER_NO_REFERENCED_ROW](#))

Message: Cannot add or update a child row: a foreign key constraint fails

- Error: [1217](#) SQLSTATE: [23000](#) ([ER_ROW_IS_REFERENCED](#))

Message: Cannot delete or update a parent row: a foreign key constraint fails

- Error: [1218](#) SQLSTATE: [08S01](#) ([ER_CONNECT_TO_MASTER](#))

Message: Error connecting to master: %s

- Error: [1219](#) SQLSTATE: [HY000](#) ([ER_QUERY_ON_MASTER](#))

Message: Error running query on master: %s

- Error: [1220](#) SQLSTATE: [HY000](#) ([ER_ERROR_WHEN_EXECUTING_COMMAND](#))

Message: Error when executing command %s: %s

- Error: [1221](#) SQLSTATE: [HY000](#) ([ER_WRONG_USAGE](#))

Message: Incorrect usage of %s and %s

- Error: [1222](#) SQLSTATE: [21000](#) ([ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT](#))

Message: The used SELECT statements have a different number of columns

- Error: [1223](#) SQLSTATE: [HY000](#) ([ER_CANT_UPDATE_WITH_READLOCK](#))

Message: Can't execute the query because you have a conflicting read lock

- Error: [1224](#) SQLSTATE: [HY000](#) ([ER_MIXING_NOT_ALLOWED](#))

Message: Mixing of transactional and non-transactional tables is disabled

- Error: [1225](#) SQLSTATE: [HY000](#) ([ER_DUP_ARGUMENT](#))

Message: Option '%s' used twice in statement

- Error: [1226](#) SQLSTATE: [42000](#) ([ER_USER_LIMIT_REACHED](#))

Message: User '%s' has exceeded the '%s' resource (current value: %ld)

- Error: [1227](#) SQLSTATE: [42000](#) ([ER_SPECIFIC_ACCESS_DENIED_ERROR](#))

Message: Access denied; you need the %s privilege for this operation

- Error: [1228](#) SQLSTATE: [HY000](#) ([ER_LOCAL_VARIABLE](#))

Message: Variable '%s' is a SESSION variable and can't be used with SET GLOBAL

- Error: [1229](#) SQLSTATE: [HY000](#) ([ER_GLOBAL_VARIABLE](#))
Message: Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL
- Error: [1230](#) SQLSTATE: [42000](#) ([ER_NO_DEFAULT](#))
Message: Variable '%s' doesn't have a default value
- Error: [1231](#) SQLSTATE: [42000](#) ([ER_WRONG_VALUE_FOR_VAR](#))
Message: Variable '%s' can't be set to the value of '%s'
- Error: [1232](#) SQLSTATE: [42000](#) ([ER_WRONG_TYPE_FOR_VAR](#))
Message: Incorrect argument type to variable '%s'
- Error: [1233](#) SQLSTATE: [HY000](#) ([ER_VAR_CANT_BE_READ](#))
Message: Variable '%s' can only be set, not read
- Error: [1234](#) SQLSTATE: [42000](#) ([ER_CANT_USE_OPTION_HERE](#))
Message: Incorrect usage/placement of '%s'
- Error: [1235](#) SQLSTATE: [42000](#) ([ER_NOT_SUPPORTED_YET](#))
Message: This version of MySQL doesn't yet support '%s'
- Error: [1236](#) SQLSTATE: [HY000](#) ([ER_MASTER_FATAL_ERROR_READING_BINLOG](#))
Message: Got fatal error %d: '%s' from master when reading data from binary log
- Error: [1237](#) SQLSTATE: [HY000](#) ([ER_SLAVE_IGNORED_TABLE](#))
Message: Slave SQL thread ignored the query because of replicate-*-table rules
- Error: [1238](#) SQLSTATE: [HY000](#) ([ER_INCORRECT_GLOBAL_LOCAL_VAR](#))
Message: Variable '%s' is a %s variable
- Error: [1239](#) SQLSTATE: [42000](#) ([ER_WRONG_FK_DEF](#))
Message: Incorrect foreign key definition for '%s': %s
- Error: [1240](#) SQLSTATE: [HY000](#) ([ER_KEY_REF_DO_NOT_MATCH_TABLE_REF](#))
Message: Key reference and table reference don't match
- Error: [1241](#) SQLSTATE: [21000](#) ([ER_OPERAND_COLUMNS](#))
Message: Operand should contain %d column(s)
- Error: [1242](#) SQLSTATE: [21000](#) ([ER_SUBQUERY_NO_1_ROW](#))
Message: Subquery returns more than 1 row
- Error: [1243](#) SQLSTATE: [HY000](#) ([ER_UNKNOWN_STMT_HANDLER](#))
Message: Unknown prepared statement handler (%.*s) given to %s
- Error: [1244](#) SQLSTATE: [HY000](#) ([ER_CORRUPT_HELP_DB](#))
Message: Help database is corrupt or does not exist
- Error: [1245](#) SQLSTATE: [HY000](#) ([ER_CYCLIC_REFERENCE](#))
Message: Cyclic reference on subqueries

- Error: [1246](#) SQLSTATE: [HY000](#) ([ER_AUTO_CONVERT](#))
Message: Converting column '%s' from %s to %s
- Error: [1247](#) SQLSTATE: [42S22](#) ([ER_ILLEGAL_REFERENCE](#))
Message: Reference '%s' not supported (%s)
- Error: [1248](#) SQLSTATE: [42000](#) ([ER_DERIVED_MUST_HAVE_ALIAS](#))
Message: Every derived table must have its own alias
- Error: [1249](#) SQLSTATE: [01000](#) ([ER_SELECT_REDUCE](#))
Message: Select %u was reduced during optimization
- Error: [1250](#) SQLSTATE: [42000](#) ([ER_TABLENAME_NOT_ALLOWED_HERE](#))
Message: Table '%s' from one of the SELECTs cannot be used in %s
- Error: [1251](#) SQLSTATE: [08004](#) ([ER_NOT_SUPPORTED_AUTH_MODE](#))
Message: Client does not support authentication protocol requested by server; consider upgrading MySQL client
- Error: [1252](#) SQLSTATE: [42000](#) ([ER_SPATIAL_CANT_HAVE_NULL](#))
Message: All parts of a SPATIAL index must be NOT NULL
- Error: [1253](#) SQLSTATE: [42000](#) ([ER_COLLATION_CHARSET_MISMATCH](#))
Message: COLLATION '%s' is not valid for CHARACTER SET '%s'
- Error: [1254](#) SQLSTATE: [HY000](#) ([ER_SLAVE_WAS_RUNNING](#))
Message: Slave is already running
- Error: [1255](#) SQLSTATE: [HY000](#) ([ER_SLAVE_WAS_NOT_RUNNING](#))
Message: Slave already has been stopped
- Error: [1256](#) SQLSTATE: [HY000](#) ([ER_TOO_BIG_FOR_UNCOMPRESS](#))
Message: Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)
- Error: [1257](#) SQLSTATE: [HY000](#) ([ER_ZLIB_Z_MEM_ERROR](#))
Message: ZLIB: Not enough memory
- Error: [1258](#) SQLSTATE: [HY000](#) ([ER_ZLIB_Z_BUF_ERROR](#))
Message: ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)
- Error: [1259](#) SQLSTATE: [HY000](#) ([ER_ZLIB_Z_DATA_ERROR](#))
Message: ZLIB: Input data corrupted
- Error: [1260](#) SQLSTATE: [HY000](#) ([ER_CUT_VALUE_GROUP_CONCAT](#))
Message: %d line(s) were cut by GROUP_CONCAT()
- Error: [1261](#) SQLSTATE: [01000](#) ([ER_WARN_TOO_FEW_RECORDS](#))
Message: Row %ld doesn't contain data for all columns
- Error: [1262](#) SQLSTATE: [01000](#) ([ER_WARN_TOO_MANY_RECORDS](#))
Message: Row %ld was truncated; it contained more data than there were input columns

- Error: [1263](#) SQLSTATE: [22004](#) ([ER_WARN_NULL_TO_NOTNULL](#))
Message: Column set to default value; NULL supplied to NOT NULL column '%s' at row %ld
- Error: [1264](#) SQLSTATE: [22003](#) ([ER_WARN_DATA_OUT_OF_RANGE](#))
Message: Out of range value for column '%s' at row %ld
- Error: [1265](#) SQLSTATE: [01000](#) ([WARN_DATA_TRUNCATED](#))
Message: Data truncated for column '%s' at row %ld
- Error: [1266](#) SQLSTATE: [HY000](#) ([ER_WARN_USING_OTHER_HANDLER](#))
Message: Using storage engine %s for table '%s'
- Error: [1267](#) SQLSTATE: [HY000](#) ([ER_CANT_AGGREGATE_2COLLATIONS](#))
Message: Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'
- Error: [1268](#) SQLSTATE: [HY000](#) ([ER_DROP_USER](#))
Message: Cannot drop one or more of the requested users
- Error: [1269](#) SQLSTATE: [HY000](#) ([ER_REVOKE_GRANTS](#))
Message: Can't revoke all privileges for one or more of the requested users
- Error: [1270](#) SQLSTATE: [HY000](#) ([ER_CANT_AGGREGATE_3COLLATIONS](#))
Message: Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'
- Error: [1271](#) SQLSTATE: [HY000](#) ([ER_CANT_AGGREGATE_NCOLLATIONS](#))
Message: Illegal mix of collations for operation '%s'
- Error: [1272](#) SQLSTATE: [HY000](#) ([ER_VARIABLE_IS_NOT_STRUCT](#))
Message: Variable '%s' is not a variable component (can't be used as XXXX.variable_name)
- Error: [1273](#) SQLSTATE: [HY000](#) ([ER_UNKNOWN_COLLATION](#))
Message: Unknown collation: '%s'
- Error: [1274](#) SQLSTATE: [HY000](#) ([ER_SLAVE_IGNORED_SSL_PARAMS](#))
Message: SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started
- Error: [1275](#) SQLSTATE: [HY000](#) ([ER_SERVER_IS_IN_SECURE_AUTH_MODE](#))
Message: Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format
- Error: [1276](#) SQLSTATE: [HY000](#) ([ER_WARN_FIELD_RESOLVED](#))
Message: Field or reference '%s%s%s%s%s' of SELECT #%d was resolved in SELECT #%d
- Error: [1277](#) SQLSTATE: [HY000](#) ([ER_BAD_SLAVE_UNTIL_COND](#))
Message: Incorrect parameter or combination of parameters for START SLAVE UNTIL
- Error: [1278](#) SQLSTATE: [HY000](#) ([ER_MISSING_SKIP_SLAVE](#))
Message: It is recommended to use --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart

- Error: [1279](#) SQLSTATE: [HY000](#) ([ER_UNTIL_COND_IGNORED](#))
Message: SQL thread is not to be started so UNTIL options are ignored
- Error: [1280](#) SQLSTATE: [42000](#) ([ER_WRONG_NAME_FOR_INDEX](#))
Message: Incorrect index name '%s'
- Error: [1281](#) SQLSTATE: [42000](#) ([ER_WRONG_NAME_FOR_CATALOG](#))
Message: Incorrect catalog name '%s'
- Error: [1282](#) SQLSTATE: [HY000](#) ([ER_WARN_QC_RESIZE](#))
Message: Query cache failed to set size %lu; new query cache size is %lu
- Error: [1283](#) SQLSTATE: [HY000](#) ([ER_BAD_FT_COLUMN](#))
Message: Column '%s' cannot be part of FULLTEXT index
- Error: [1284](#) SQLSTATE: [HY000](#) ([ER_UNKNOWN_KEY_CACHE](#))
Message: Unknown key cache '%s'
- Error: [1285](#) SQLSTATE: [HY000](#) ([ER_WARN_HOSTNAME_WONT_WORK](#))
Message: MySQL is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work
- Error: [1286](#) SQLSTATE: [42000](#) ([ER_UNKNOWN_STORAGE_ENGINE](#))
Message: Unknown table engine '%s'
- Error: [1287](#) SQLSTATE: [HY000](#) ([ER_WARN_DEPRECATED_SYNTAX](#))
Message: '%s' is deprecated; use '%s' instead
- Error: [1288](#) SQLSTATE: [HY000](#) ([ER_NON_UPDATABLE_TABLE](#))
Message: The target table %s of the %s is not updatable
- Error: [1289](#) SQLSTATE: [HY000](#) ([ER_FEATURE_DISABLED](#))
Message: The '%s' feature is disabled; you need MySQL built with '%s' to have it working
- Error: [1290](#) SQLSTATE: [HY000](#) ([ER_OPTION_PREVENTS_STATEMENT](#))
Message: The MySQL server is running with the %s option so it cannot execute this statement
- Error: [1291](#) SQLSTATE: [HY000](#) ([ER_DUPLICATED_VALUE_IN_TYPE](#))
Message: Column '%s' has duplicated value '%s' in %s
- Error: [1292](#) SQLSTATE: [22007](#) ([ER_TRUNCATED_WRONG_VALUE](#))
Message: Truncated incorrect %s value: '%s'
- Error: [1293](#) SQLSTATE: [HY000](#) ([ER_TOO_MUCH_AUTO_TIMESTAMP_COLS](#))
Message: Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause
- Error: [1294](#) SQLSTATE: [HY000](#) ([ER_INVALID_ON_UPDATE](#))
Message: Invalid ON UPDATE clause for '%s' column
- Error: [1295](#) SQLSTATE: [HY000](#) ([ER_UNSUPPORTED_PS](#))

Message: This command is not supported in the prepared statement protocol yet

- Error: **1296** SQLSTATE: **HY000** (**ER_GET_ERRMSG**)

Message: Got error %d '%s' from %s

- Error: **1297** SQLSTATE: **HY000** (**ER_GET_TEMPORARY_ERRMSG**)

Message: Got temporary error %d '%s' from %s

- Error: **1298** SQLSTATE: **HY000** (**ER_UNKNOWN_TIME_ZONE**)

Message: Unknown or incorrect time zone: '%s'

- Error: **1299** SQLSTATE: **HY000** (**ER_WARN_INVALID_TIMESTAMP**)

Message: Invalid TIMESTAMP value in column '%s' at row %ld

- Error: **1300** SQLSTATE: **HY000** (**ER_INVALID_CHARACTER_STRING**)

Message: Invalid %s character string: '%s'

- Error: **1301** SQLSTATE: **HY000** (**ER_WARN_ALLOWED_PACKET_OVERFLOWED**)

Message: Result of %s() was larger than max_allowed_packet (%ld) - truncated

- Error: **1302** SQLSTATE: **HY000** (**ER_CONFLICTING_DECLARATIONS**)

Message: Conflicting declarations: '%s%s' and '%s%s'

- Error: **1303** SQLSTATE: **2F003** (**ER_SP_NO_RECURSIVE_CREATE**)

Message: Can't create a %s from within another stored routine

- Error: **1304** SQLSTATE: **42000** (**ER_SP_ALREADY_EXISTS**)

Message: %s %s already exists

- Error: **1305** SQLSTATE: **42000** (**ER_SP_DOES_NOT_EXIST**)

Message: %s %s does not exist

- Error: **1306** SQLSTATE: **HY000** (**ER_SP_DROP_FAILED**)

Message: Failed to DROP %s %s

- Error: **1307** SQLSTATE: **HY000** (**ER_SP_STORE_FAILED**)

Message: Failed to CREATE %s %s

- Error: **1308** SQLSTATE: **42000** (**ER_SP_LILABEL_MISMATCH**)

Message: %s with no matching label: %s

- Error: **1309** SQLSTATE: **42000** (**ER_SP_LABEL_REDEFINE**)

Message: Redefining label %s

- Error: **1310** SQLSTATE: **42000** (**ER_SP_LABEL_MISMATCH**)

Message: End-label %s without match

- Error: **1311** SQLSTATE: **01000** (**ER_SP_UNINIT_VAR**)

Message: Referring to uninitialized variable %s

- Error: **1312** SQLSTATE: **0A000** (**ER_SP_BADSELECT**)
Message: PROCEDURE %s can't return a result set in the given context
- Error: **1313** SQLSTATE: **42000** (**ER_SP_BADRETURN**)
Message: RETURN is only allowed in a FUNCTION
- Error: **1314** SQLSTATE: **0A000** (**ER_SP_BADSTATEMENT**)
Message: %s is not allowed in stored procedures
- Error: **1315** SQLSTATE: **42000** (**ER_UPDATE_LOG_DEPRECATED_IGNORED**)
Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been ignored
- Error: **1316** SQLSTATE: **42000** (**ER_UPDATE_LOG_DEPRECATED_TRANSLATED**)
Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been translated to SET SQL_LOG_BIN
- Error: **1317** SQLSTATE: **70100** (**ER_QUERY_INTERRUPTED**)
Message: Query execution was interrupted
- Error: **1318** SQLSTATE: **42000** (**ER_SP_WRONG_NO_OF_ARGS**)
Message: Incorrect number of arguments for %s %s; expected %u, got %u
- Error: **1319** SQLSTATE: **42000** (**ER_SP_COND_MISMATCH**)
Message: Undefined CONDITION: %s
- Error: **1320** SQLSTATE: **42000** (**ER_SP_NORETURN**)
Message: No RETURN found in FUNCTION %s
- Error: **1321** SQLSTATE: **2F005** (**ER_SP_NORETURNEND**)
Message: FUNCTION %s ended without RETURN
- Error: **1322** SQLSTATE: **42000** (**ER_SP_BAD_CURSOR_QUERY**)
Message: Cursor statement must be a SELECT
- Error: **1323** SQLSTATE: **42000** (**ER_SP_BAD_CURSOR_SELECT**)
Message: Cursor SELECT must not have INTO
- Error: **1324** SQLSTATE: **42000** (**ER_SP_CURSOR_MISMATCH**)
Message: Undefined CURSOR: %s
- Error: **1325** SQLSTATE: **24000** (**ER_SP_CURSOR_ALREADY_OPEN**)
Message: Cursor is already open
- Error: **1326** SQLSTATE: **24000** (**ER_SP_CURSOR_NOT_OPEN**)
Message: Cursor is not open
- Error: **1327** SQLSTATE: **42000** (**ER_SP_UNDECLARED_VAR**)
Message: Undeclared variable: %s
- Error: **1328** SQLSTATE: **HY000** (**ER_SP_WRONG_NO_OF_FETCH_ARGS**)

Message: Incorrect number of FETCH variables

- Error: **1329** SQLSTATE: **02000** (**ER_SP_FETCH_NO_DATA**)

Message: No data - zero rows fetched, selected, or processed

- Error: **1330** SQLSTATE: **42000** (**ER_SP_DUP_PARAM**)

Message: Duplicate parameter: %s

- Error: **1331** SQLSTATE: **42000** (**ER_SP_DUP_VAR**)

Message: Duplicate variable: %s

- Error: **1332** SQLSTATE: **42000** (**ER_SP_DUP_COND**)

Message: Duplicate condition: %s

- Error: **1333** SQLSTATE: **42000** (**ER_SP_DUP_CURS**)

Message: Duplicate cursor: %s

- Error: **1334** SQLSTATE: **HY000** (**ER_SP_CANT_ALTER**)

Message: Failed to ALTER %s %s

- Error: **1335** SQLSTATE: **0A000** (**ER_SP_SUBSELECT_NYI**)

Message: Subquery value not supported

- Error: **1336** SQLSTATE: **0A000** (**ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG**)

Message: %s is not allowed in stored function or trigger

- Error: **1337** SQLSTATE: **42000** (**ER_SP_VARCOND_AFTER_CURSHNDLR**)

Message: Variable or condition declaration after cursor or handler declaration

- Error: **1338** SQLSTATE: **42000** (**ER_SP_CURSOR_AFTER_HANDLER**)

Message: Cursor declaration after handler declaration

- Error: **1339** SQLSTATE: **20000** (**ER_SP_CASE_NOT_FOUND**)

Message: Case not found for CASE statement

- Error: **1340** SQLSTATE: **HY000** (**ER_FPARSER_TOO_BIG_FILE**)

Message: Configuration file '%s' is too big

- Error: **1341** SQLSTATE: **HY000** (**ER_FPARSER_BAD_HEADER**)

Message: Malformed file type header in file '%s'

- Error: **1342** SQLSTATE: **HY000** (**ER_FPARSER_EOF_IN_COMMENT**)

Message: Unexpected end of file while parsing comment '%s'

- Error: **1343** SQLSTATE: **HY000** (**ER_FPARSER_ERROR_IN_PARAMETER**)

Message: Error while parsing parameter '%s' (line: '%s')

- Error: **1344** SQLSTATE: **HY000** (**ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER**)

Message: Unexpected end of file while skipping unknown parameter '%s'

- Error: [1345](#) SQLSTATE: [HY000](#) ([ER_VIEW_NO_EXPLAIN](#))
Message: EXPLAIN/SHOW can not be issued; lacking privileges for underlying table
- Error: [1346](#) SQLSTATE: [HY000](#) ([ER_FRM_UNKNOWN_TYPE](#))
Message: File '%s' has unknown type '%s' in its header
- Error: [1347](#) SQLSTATE: [HY000](#) ([ER_WRONG_OBJECT](#))
Message: '%s.%s' is not %s
- Error: [1348](#) SQLSTATE: [HY000](#) ([ER_NONUPDATEABLE_COLUMN](#))
Message: Column '%s' is not updatable
- Error: [1349](#) SQLSTATE: [HY000](#) ([ER_VIEW_SELECT_DERIVED](#))
Message: View's SELECT contains a subquery in the FROM clause
- Error: [1350](#) SQLSTATE: [HY000](#) ([ER_VIEW_SELECT_CLAUSE](#))
Message: View's SELECT contains a '%s' clause
- Error: [1351](#) SQLSTATE: [HY000](#) ([ER_VIEW_SELECT_VARIABLE](#))
Message: View's SELECT contains a variable or parameter
- Error: [1352](#) SQLSTATE: [HY000](#) ([ER_VIEW_SELECT_TMPTABLE](#))
Message: View's SELECT refers to a temporary table '%s'
- Error: [1353](#) SQLSTATE: [HY000](#) ([ER_VIEW_WRONG_LIST](#))
Message: View's SELECT and view's field list have different column counts
- Error: [1354](#) SQLSTATE: [HY000](#) ([ER_WARN_VIEW_MERGE](#))
Message: View merge algorithm can't be used here for now (assumed undefined algorithm)
- Error: [1355](#) SQLSTATE: [HY000](#) ([ER_WARN_VIEW_WITHOUT_KEY](#))
Message: View being updated does not have complete key of underlying table in it
- Error: [1356](#) SQLSTATE: [HY000](#) ([ER_VIEW_INVALID](#))
Message: View '%s.%s' references invalid table(s) or column(s) or function(s) or definer invoker of view lack rights to use them
- Error: [1357](#) SQLSTATE: [HY000](#) ([ER_SP_NO_DROP_SP](#))
Message: Can't drop or alter a %s from within another stored routine
- Error: [1358](#) SQLSTATE: [HY000](#) ([ER_SP_GOTO_IN_HNDLR](#))
Message: GOTO is not allowed in a stored procedure handler
- Error: [1359](#) SQLSTATE: [HY000](#) ([ER_TRG_ALREADY_EXISTS](#))
Message: Trigger already exists
- Error: [1360](#) SQLSTATE: [HY000](#) ([ER_TRG_DOES_NOT_EXIST](#))
Message: Trigger does not exist
- Error: [1361](#) SQLSTATE: [HY000](#) ([ER_TRG_ON_VIEW_OR_TEMP_TABLE](#))
Message: Trigger's '%s' is view or temporary table

- Error: [1362](#) SQLSTATE: [HY000](#) ([ER_TRG_CANT_CHANGE_ROW](#))
Message: Updating of %s row is not allowed in %strigger
- Error: [1363](#) SQLSTATE: [HY000](#) ([ER_TRG_NO_SUCH_ROW_IN_TRG](#))
Message: There is no %s row in %s trigger
- Error: [1364](#) SQLSTATE: [HY000](#) ([ER_NO_DEFAULT_FOR_FIELD](#))
Message: Field '%s' doesn't have a default value
- Error: [1365](#) SQLSTATE: [22012](#) ([ER_DIVISION_BY_ZERO](#))
Message: Division by 0
- Error: [1366](#) SQLSTATE: [HY000](#) ([ER_TRUNCATED_WRONG_VALUE_FOR_FIELD](#))
Message: Incorrect %s value: '%s' for column '%s' at row %ld
- Error: [1367](#) SQLSTATE: [22007](#) ([ER_ILLEGAL_VALUE_FOR_TYPE](#))
Message: Illegal %s '%s' value found during parsing
- Error: [1368](#) SQLSTATE: [HY000](#) ([ER_VIEW_NONUPD_CHECK](#))
Message: CHECK OPTION on non-updatable view '%s.%s'
- Error: [1369](#) SQLSTATE: [HY000](#) ([ER_VIEW_CHECK_FAILED](#))
Message: CHECK OPTION failed '%s.%s'
- Error: [1370](#) SQLSTATE: [42000](#) ([ER_PROCACCESS_DENIED_ERROR](#))
Message: %s command denied to user '%s'@'%s' for routine '%s'
- Error: [1371](#) SQLSTATE: [HY000](#) ([ER_RELAY_LOG_FAIL](#))
Message: Failed purging old relay logs: %s
- Error: [1372](#) SQLSTATE: [HY000](#) ([ER_PASSWD_LENGTH](#))
Message: Password hash should be a %d-digit hexadecimal number
- Error: [1373](#) SQLSTATE: [HY000](#) ([ER_UNKNOWN_TARGET_BINLOG](#))
Message: Target log not found in binlog index
- Error: [1374](#) SQLSTATE: [HY000](#) ([ER_IO_ERR_LOG_INDEX_READ](#))
Message: I/O error reading log index file
- Error: [1375](#) SQLSTATE: [HY000](#) ([ER_BINLOG_PURGE_PROHIBITED](#))
Message: Server configuration does not permit binlog purge
- Error: [1376](#) SQLSTATE: [HY000](#) ([ER_FSEEK_FAIL](#))
Message: Failed on fseek()
- Error: [1377](#) SQLSTATE: [HY000](#) ([ER_BINLOG_PURGE_FATAL_ERR](#))
Message: Fatal error during log purge
- Error: [1378](#) SQLSTATE: [HY000](#) ([ER_LOG_IN_USE](#))
Message: A purgeable log is in use, will not purge

- Error: [1379](#) SQLSTATE: [HY000](#) ([ER_LOG_PURGE_UNKNOWN_ERR](#))
Message: Unknown error during log purge
- Error: [1380](#) SQLSTATE: [HY000](#) ([ER_RELAY_LOG_INIT](#))
Message: Failed initializing relay log position: %s
- Error: [1381](#) SQLSTATE: [HY000](#) ([ER_NO_BINARY_LOGGING](#))
Message: You are not using binary logging
- Error: [1382](#) SQLSTATE: [HY000](#) ([ER_RESERVED_SYNTAX](#))
Message: The '%s' syntax is reserved for purposes internal to the MySQL server
- Error: [1383](#) SQLSTATE: [HY000](#) ([ER_WSAS_FAILED](#))
Message: WSAStartup Failed
- Error: [1384](#) SQLSTATE: [HY000](#) ([ER_DIFF_GROUPS_PROC](#))
Message: Can't handle procedures with different groups yet
- Error: [1385](#) SQLSTATE: [HY000](#) ([ER_NO_GROUP_FOR_PROC](#))
Message: Select must have a group with this procedure
- Error: [1386](#) SQLSTATE: [HY000](#) ([ER_ORDER_WITH_PROC](#))
Message: Can't use ORDER clause with this procedure
- Error: [1387](#) SQLSTATE: [HY000](#) ([ER_LOGGING_PROHIBIT_CHANGING_OF](#))
Message: Binary logging and replication forbid changing the global server %s
- Error: [1388](#) SQLSTATE: [HY000](#) ([ER_NO_FILE_MAPPING](#))
Message: Can't map file: %s, errno: %d
- Error: [1389](#) SQLSTATE: [HY000](#) ([ER_WRONG_MAGIC](#))
Message: Wrong magic in %s
- Error: [1390](#) SQLSTATE: [HY000](#) ([ER_PS_MANY_PARAM](#))
Message: Prepared statement contains too many placeholders
- Error: [1391](#) SQLSTATE: [HY000](#) ([ER_KEY_PART_0](#))
Message: Key part '%s' length cannot be 0
- Error: [1392](#) SQLSTATE: [HY000](#) ([ER_VIEW_CHECKSUM](#))
Message: View text checksum failed
- Error: [1393](#) SQLSTATE: [HY000](#) ([ER_VIEW_MULTIUPDATE](#))
Message: Can not modify more than one base table through a join view '%s.%s'
- Error: [1394](#) SQLSTATE: [HY000](#) ([ER_VIEW_NO_INSERT_FIELD_LIST](#))
Message: Can not insert into join view '%s.%s' without fields list
- Error: [1395](#) SQLSTATE: [HY000](#) ([ER_VIEW_DELETE_MERGE_VIEW](#))
Message: Can not delete from join view '%s.%s'

- Error: [1396](#) SQLSTATE: [HY000](#) ([ER_CANNOT_USER](#))
Message: Operation %s failed for %s
- Error: [1397](#) SQLSTATE: [XAE04](#) ([ER_XAER_NOTA](#))
Message: XAER_NOTA: Unknown XID
- Error: [1398](#) SQLSTATE: [XAE05](#) ([ER_XAER_INVAL](#))
Message: XAER_INVAL: Invalid arguments (or unsupported command)
- Error: [1399](#) SQLSTATE: [XAE07](#) ([ER_XAER_RMFAIL](#))
Message: XAER_RMFAIL: The command cannot be executed when global transaction is in the %s state
- Error: [1400](#) SQLSTATE: [XAE09](#) ([ER_XAER_OUTSIDE](#))
Message: XAER_OUTSIDE: Some work is done outside global transaction
- Error: [1401](#) SQLSTATE: [XAE03](#) ([ER_XAER_RMERR](#))
Message: XAER_RMERR: Fatal error occurred in the transaction branch - check your data for consistency
- Error: [1402](#) SQLSTATE: [XA100](#) ([ER_XA_RBROLLBACK](#))
Message: XA_RBROLLBACK: Transaction branch was rolled back
- Error: [1403](#) SQLSTATE: [42000](#) ([ER_NONEXISTING_PROC_GRANT](#))
Message: There is no such grant defined for user '%s' on host '%s' on routine '%s'
- Error: [1404](#) SQLSTATE: [HY000](#) ([ER_PROC_AUTO_GRANT_FAIL](#))
Message: Failed to grant EXECUTE and ALTER ROUTINE privileges
- Error: [1405](#) SQLSTATE: [HY000](#) ([ER_PROC_AUTO_REVOKE_FAIL](#))
Message: Failed to revoke all privileges to dropped routine
- Error: [1406](#) SQLSTATE: [22001](#) ([ER_DATA_TOO_LONG](#))
Message: Data too long for column '%s' at row %ld
- Error: [1407](#) SQLSTATE: [42000](#) ([ER_SP_BAD_SQLSTATE](#))
Message: Bad SQLSTATE: '%s'
- Error: [1408](#) SQLSTATE: [HY000](#) ([ER_STARTUP](#))
Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d %s
- Error: [1409](#) SQLSTATE: [HY000](#) ([ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR](#))
Message: Can't load value from file with fixed size rows to variable
- Error: [1410](#) SQLSTATE: [42000](#) ([ER_CANT_CREATE_USER_WITH_GRANT](#))
Message: You are not allowed to create a user with GRANT
- Error: [1411](#) SQLSTATE: [HY000](#) ([ER_WRONG_VALUE_FOR_TYPE](#))
Message: Incorrect %s value: '%s' for function %s
- Error: [1412](#) SQLSTATE: [HY000](#) ([ER_TABLE_DEF_CHANGED](#))
Message: Table definition has changed, please retry transaction

- Error: [1413](#) SQLSTATE: [42000](#) ([ER_SP_DUP_HANDLER](#))
Message: Duplicate handler declared in the same block
- Error: [1414](#) SQLSTATE: [42000](#) ([ER_SP_NOT_VAR_ARG](#))
Message: OUT or INOUT argument %d for routine %s is not a variable or NEW pseudo-variable in BEFORE trigger
- Error: [1415](#) SQLSTATE: [0A000](#) ([ER_SP_NO_RETSET](#))
Message: Not allowed to return a result set from a %s
- Error: [1416](#) SQLSTATE: [22003](#) ([ER_CANT_CREATE_GEOMETRY_OBJECT](#))
Message: Cannot get geometry object from data you send to the GEOMETRY field
- Error: [1417](#) SQLSTATE: [HY000](#) ([ER_FAILED_ROUTINE_BREAK_BINLOG](#))
Message: A routine failed and has neither NO SQL nor READS SQL DATA in its declaration and binary logging is enabled; if non-transactional tables were updated, the binary log will miss their changes
- Error: [1418](#) SQLSTATE: [HY000](#) ([ER_BINLOG_UNSAFE_ROUTINE](#))
Message: This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
- Error: [1419](#) SQLSTATE: [HY000](#) ([ER_BINLOG_CREATE_ROUTINE_NEED_SUPER](#))
Message: You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
- Error: [1420](#) SQLSTATE: [HY000](#) ([ER_EXEC_STMT_WITH_OPEN_CURSOR](#))
Message: You can't execute a prepared statement which has an open cursor associated with it. Reset the statement to re-execute it.
- Error: [1421](#) SQLSTATE: [HY000](#) ([ER_STMT_HAS_NO_OPEN_CURSOR](#))
Message: The statement (%lu) has no open cursor.
- Error: [1422](#) SQLSTATE: [HY000](#) ([ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG](#))
Message: Explicit or implicit commit is not allowed in stored function or trigger.
- Error: [1423](#) SQLSTATE: [HY000](#) ([ER_NO_DEFAULT_FOR_VIEW_FIELD](#))
Message: Field of view '%s.%s' underlying table doesn't have a default value
- Error: [1424](#) SQLSTATE: [HY000](#) ([ER_SP_NO_RECURSION](#))
Message: Recursive stored functions and triggers are not allowed.
- Error: [1425](#) SQLSTATE: [42000](#) ([ER_TOO_BIG_SCALE](#))
Message: Too big scale %d specified for column '%s'. Maximum is %d.
- Error: [1426](#) SQLSTATE: [42000](#) ([ER_TOO_BIG_PRECISION](#))
Message: Too big precision %d specified for column '%s'. Maximum is %d.
- Error: [1427](#) SQLSTATE: [42000](#) ([ER_M_BIGGER_THAN_D](#))
Message: For float(M,D), double(M,D) or decimal(M,D), M must be >= D (column '%s').
- Error: [1428](#) SQLSTATE: [HY000](#) ([ER_WRONG_LOCK_OF_SYSTEM_TABLE](#))
Message: You can't combine write-locking of system tables with other tables or lock types

- Error: **1429** SQLSTATE: **HY000** (**ER_CONNECT_TO_FOREIGN_DATA_SOURCE**)
Message: Unable to connect to foreign data source: %s
- Error: **1430** SQLSTATE: **HY000** (**ER_QUERY_ON_FOREIGN_DATA_SOURCE**)
Message: There was a problem processing the query on the foreign data source. Data source error: %s
- Error: **1431** SQLSTATE: **HY000** (**ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST**)
Message: The foreign data source you are trying to reference does not exist. Data source error: %s
- Error: **1432** SQLSTATE: **HY000** (**ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE**)
Message: Can't create federated table. The data source connection string '%s' is not in the correct format
- Error: **1433** SQLSTATE: **HY000** (**ER_FOREIGN_DATA_STRING_INVALID**)
Message: The data source connection string '%s' is not in the correct format
- Error: **1434** SQLSTATE: **HY000** (**ER_CANT_CREATE_FEDERATED_TABLE**)
Message: Can't create federated table. Foreign data src error: %s
- Error: **1435** SQLSTATE: **HY000** (**ER_TRG_IN_WRONG_SCHEMA**)
Message: Trigger in wrong schema
- Error: **1436** SQLSTATE: **HY000** (**ER_STACK_OVERRUN_NEED_MORE**)
Message: Thread stack overrun: %ld bytes used of a %ld byte stack, and %ld bytes needed. Use 'mysqld -O thread_stack=#' to specify a bigger stack.
- Error: **1437** SQLSTATE: **42000** (**ER_TOO_LONG_BODY**)
Message: Routine body for '%s' is too long
- Error: **1438** SQLSTATE: **HY000** (**ER_WARN_CANT_DROP_DEFAULT_KEYCACHE**)
Message: Cannot drop default keycache
- Error: **1439** SQLSTATE: **42000** (**ER_TOO_BIG_DISPLAYWIDTH**)
Message: Display width out of range for column '%s' (max = %d)
- Error: **1440** SQLSTATE: **XAE08** (**ER_XAER_DUPID**)
Message: XAER_DUPID: The XID already exists
- Error: **1441** SQLSTATE: **22008** (**ER_DATETIME_FUNCTION_OVERFLOW**)
Message: Datetime function: %s field overflow
- Error: **1442** SQLSTATE: **HY000** (**ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG**)
Message: Can't update table '%s' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.
- Error: **1443** SQLSTATE: **HY000** (**ER_VIEW_PREVENT_UPDATE**)
Message: The definition of table '%s' prevents operation %s on table '%s'.
- Error: **1444** SQLSTATE: **HY000** (**ER_PS_NO_RECURSION**)
Message: The prepared statement contains a stored routine call that refers to that same statement. It's not allowed to execute a prepared statement in such a recursive manner

- Error: [1445](#) SQLSTATE: [HY000](#) ([ER_SP_CANT_SET_AUTOCOMMIT](#))
Message: Not allowed to set autocommit from a stored function or trigger
- Error: [1446](#) SQLSTATE: [HY000](#) ([ER_MALFORMED_DEFINER](#))
Message: Definer is not fully qualified
- Error: [1447](#) SQLSTATE: [HY000](#) ([ER_VIEW_FRM_NO_USER](#))
Message: View '%s'.'%s' has no definer information (old table format). Current user is used as definer. Please recreate the view!
- Error: [1448](#) SQLSTATE: [HY000](#) ([ER_VIEW_OTHER_USER](#))
Message: You need the SUPER privilege for creation view with '%s'@'%s' definer
- Error: [1449](#) SQLSTATE: [HY000](#) ([ER_NO_SUCH_USER](#))
Message: There is no '%s'@'%s' registered
- Error: [1450](#) SQLSTATE: [HY000](#) ([ER_FORBID_SCHEMA_CHANGE](#))
Message: Changing schema from '%s' to '%s' is not allowed.
- Error: [1451](#) SQLSTATE: [23000](#) ([ER_ROW_IS_REFERENCED_2](#))
Message: Cannot delete or update a parent row: a foreign key constraint fails (%s)
- Error: [1452](#) SQLSTATE: [23000](#) ([ER_NO_REFERENCED_ROW_2](#))
Message: Cannot add or update a child row: a foreign key constraint fails (%s)
- Error: [1453](#) SQLSTATE: [42000](#) ([ER_SP_BAD_VAR_SHADOW](#))
Message: Variable '%s' must be quoted with `...`, or renamed
- Error: [1454](#) SQLSTATE: [HY000](#) ([ER_TRG_NO_DEFINER](#))
Message: No definer attribute for trigger '%s'.'%s'. The trigger will be activated under the authorization of the invoker, which may have insufficient privileges. Please recreate the trigger.
- Error: [1455](#) SQLSTATE: [HY000](#) ([ER_OLD_FILE_FORMAT](#))
Message: '%s' has an old format, you should re-create the '%s' object(s)
- Error: [1456](#) SQLSTATE: [HY000](#) ([ER_SP_RECURSION_LIMIT](#))
Message: Recursive limit %d (as set by the max_sp_recursion_depth variable) was exceeded for routine %s
- Error: [1457](#) SQLSTATE: [HY000](#) ([ER_SP_PROC_TABLE_CORRUPT](#))
Message: Failed to load routine %s. The table mysql.proc is missing, corrupt, or contains bad data (internal code %d)
- Error: [1458](#) SQLSTATE: [42000](#) ([ER_SP_WRONG_NAME](#))
Message: Incorrect routine name '%s'
- Error: [1459](#) SQLSTATE: [HY000](#) ([ER_TABLE_NEEDS_UPGRADE](#))
Message: Table upgrade required. Please do "REPAIR TABLE `%s`" to fix it!
- Error: [1460](#) SQLSTATE: [42000](#) ([ER_SP_NO_AGGREGATE](#))
Message: AGGREGATE is not supported for stored functions
- Error: [1461](#) SQLSTATE: [42000](#) ([ER_MAX_PREPARED_STMT_COUNT_REACHED](#))

Message: Can't create more than max_prepared_stmt_count statements (current value: %lu)

- Error: [1462](#) SQLSTATE: [HY000](#) ([ER_VIEW_RECURSIVE](#))

Message: '%s'. '%s' contains view recursion

- Error: [1463](#) SQLSTATE: [42000](#) ([ER_NON_GROUPING_FIELD_USED](#))

Message: non-grouping field '%s' is used in %s clause

- Error: [1464](#) SQLSTATE: [HY000](#) ([ER_TABLE_CANT_HANDLE_SPKEYS](#))

Message: The used table type doesn't support SPATIAL indexes

- Error: [1465](#) SQLSTATE: [HY000](#) ([ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA](#))

Message: Triggers can not be created on system tables

- Error: [1466](#) SQLSTATE: [HY000](#) ([ER_REMOVED_SPACES](#))

Message: Leading spaces are removed from name '%s'

- Error: [1467](#) SQLSTATE: [HY000](#) ([ER_AUTOINC_READ_FAILED](#))

Message: Failed to read auto-increment value from storage engine

- Error: [1468](#) SQLSTATE: [HY000](#) ([ER_USERNAME](#))

Message: user name

- Error: [1469](#) SQLSTATE: [HY000](#) ([ER_HOSTNAME](#))

Message: host name

- Error: [1470](#) SQLSTATE: [HY000](#) ([ER_WRONG_STRING_LENGTH](#))

Message: String '%s' is too long for %s (should be no longer than %d)

- Error: [1471](#) SQLSTATE: [HY000](#) ([ER_NON_INSERTABLE_TABLE](#))

Message: The target table %s of the %s is not insertable-into

- Error: [1472](#) SQLSTATE: [HY000](#) ([ER_ADMIN_WRONG_MRG_TABLE](#))

Message: Table '%s' is differently defined or of non-MyISAM type or doesn't exist

- Error: [1473](#) SQLSTATE: [HY000](#) ([ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT](#))

Message: Too high level of nesting for select

- Error: [1474](#) SQLSTATE: [HY000](#) ([ER_FOREIGN_SERVER_EXISTS](#))

Message: The foreign server, %s, you are trying to create already exists.

- Error: [1475](#) SQLSTATE: [HY000](#) ([ER_FOREIGN_SERVER_DOESNT_EXIST](#))

Message: The foreign server name you are trying to reference does not exist. Data source error: %s

- Error: [1476](#) SQLSTATE: [HY000](#) ([ER_ILLEGAL_HA_CREATE_OPTION](#))

Message: Table storage engine '%s' does not support the create option '%s'

- Error: [1477](#) SQLSTATE: [HY000](#) ([ER_PARTITION_REQUIRES_VALUES_ERROR](#))

Message: %s PARTITIONING requires definition of VALUES %s for each partition

- Error: [1478](#) SQLSTATE: [HY000](#) ([ER_PARTITION_WRONG_VALUES_ERROR](#))
Message: Only %s PARTITIONING can use VALUES %s in partition definition
- Error: [1479](#) SQLSTATE: [HY000](#) ([ER_PARTITION_MAXVALUE_ERROR](#))
Message: MAXVALUE can only be used in last partition definition
- Error: [1480](#) SQLSTATE: [HY000](#) ([ER_PARTITION_SUBPARTITION_ERROR](#))
Message: Subpartitions can only be hash partitions and by key
- Error: [1481](#) SQLSTATE: [HY000](#) ([ER_PARTITION_SUBPART_MIX_ERROR](#))
Message: Must define subpartitions on all partitions if on one partition
- Error: [1482](#) SQLSTATE: [HY000](#) ([ER_PARTITION_WRONG_NO_PART_ERROR](#))
Message: Wrong number of partitions defined, mismatch with previous setting
- Error: [1483](#) SQLSTATE: [HY000](#) ([ER_PARTITION_WRONG_NO_SUBPART_ERROR](#))
Message: Wrong number of subpartitions defined, mismatch with previous setting
- Error: [1484](#) SQLSTATE: [HY000](#) ([ER_CONST_EXPR_IN_PARTITION_FUNC_ERROR](#))
Message: Constant/Random expression in (sub)partitioning function is not allowed
- Error: [1485](#) SQLSTATE: [HY000](#) ([ER_NO_CONST_EXPR_IN_RANGE_OR_LIST_ERROR](#))
Message: Expression in RANGE/LIST VALUES must be constant
- Error: [1486](#) SQLSTATE: [HY000](#) ([ER_FIELD_NOT_FOUND_PART_ERROR](#))
Message: Field in list of fields for partition function not found in table
- Error: [1487](#) SQLSTATE: [HY000](#) ([ER_LIST_OF_FIELDS_ONLY_IN_HASH_ERROR](#))
Message: List of fields is only allowed in KEY partitions
- Error: [1488](#) SQLSTATE: [HY000](#) ([ER_INCONSISTENT_PARTITION_INFO_ERROR](#))
Message: The partition info in the frm file is not consistent with what can be written into the frm file
- Error: [1489](#) SQLSTATE: [HY000](#) ([ER_PARTITION_FUNC_NOT_ALLOWED_ERROR](#))
Message: The %s function returns the wrong type
- Error: [1490](#) SQLSTATE: [HY000](#) ([ER_PARTITIONS_MUST_BE_DEFINED_ERROR](#))
Message: For %s partitions each partition must be defined
- Error: [1491](#) SQLSTATE: [HY000](#) ([ER_RANGE_NOT_INCREASING_ERROR](#))
Message: VALUES LESS THAN value must be strictly increasing for each partition
- Error: [1492](#) SQLSTATE: [HY000](#) ([ER_INCONSISTENT_TYPE_OF_FUNCTIONS_ERROR](#))
Message: VALUES value must be of same type as partition function
- Error: [1493](#) SQLSTATE: [HY000](#) ([ER_MULTIPLE_DEF_CONST_IN_LIST_PART_ERROR](#))
Message: Multiple definition of same constant in list partitioning
- Error: [1494](#) SQLSTATE: [HY000](#) ([ER_PARTITION_ENTRY_ERROR](#))
Message: Partitioning can not be used stand-alone in query

- Error: [1495](#) SQLSTATE: [HY000](#) ([ER_MIX_HANDLER_ERROR](#))
Message: The mix of handlers in the partitions is not allowed in this version of MySQL
- Error: [1496](#) SQLSTATE: [HY000](#) ([ER_PARTITION_NOT_DEFINED_ERROR](#))
Message: For the partitioned engine it is necessary to define all %s
- Error: [1497](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_PARTITIONS_ERROR](#))
Message: Too many partitions (including subpartitions) were defined
- Error: [1498](#) SQLSTATE: [HY000](#) ([ER_SUBPARTITION_ERROR](#))
Message: It is only possible to mix RANGE/LIST partitioning with HASH/KEY partitioning for subpartitioning
- Error: [1499](#) SQLSTATE: [HY000](#) ([ER_CANT_CREATE_HANDLER_FILE](#))
Message: Failed to create specific handler file
- Error: [1500](#) SQLSTATE: [HY000](#) ([ER_BLOB_FIELD_IN_PART_FUNC_ERROR](#))
Message: A BLOB field is not allowed in partition function
- Error: [1501](#) SQLSTATE: [HY000](#) ([ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF](#))
Message: A %s must include all columns in the table's partitioning function
- Error: [1502](#) SQLSTATE: [HY000](#) ([ER_NO_PARTS_ERROR](#))
Message: Number of %s = 0 is not an allowed value
- Error: [1503](#) SQLSTATE: [HY000](#) ([ER_PARTITION_MGMT_ON_NONPARTITIONED](#))
Message: Partition management on a not partitioned table is not possible
- Error: [1504](#) SQLSTATE: [HY000](#) ([ER_FOREIGN_KEY_ON_PARTITIONED](#))
Message: Foreign key condition is not yet supported in conjunction with partitioning
- Error: [1505](#) SQLSTATE: [HY000](#) ([ER_DROP_PARTITION_NON_EXISTENT](#))
Message: Error in list of partitions to %s
- Error: [1506](#) SQLSTATE: [HY000](#) ([ER_DROP_LAST_PARTITION](#))
Message: Cannot remove all partitions, use DROP TABLE instead
- Error: [1507](#) SQLSTATE: [HY000](#) ([ER_COALESCE_ONLY_ON_HASH_PARTITION](#))
Message: COALESCE PARTITION can only be used on HASH/KEY partitions
- Error: [1508](#) SQLSTATE: [HY000](#) ([ER_REORG_HASH_ONLY_ON_SAME_NO](#))
Message: REORGANISE PARTITION can only be used to reorganise partitions not to change their numbers
- Error: [1509](#) SQLSTATE: [HY000](#) ([ER_REORG_NO_PARAM_ERROR](#))
Message: REORGANISE PARTITION without parameters can only be used on auto-partitioned tables using HASH PARTITIONS
- Error: [1510](#) SQLSTATE: [HY000](#) ([ER_ONLY_ON_RANGE_LIST_PARTITION](#))
Message: %s PARTITION can only be used on RANGE/LIST partitions
- Error: [1511](#) SQLSTATE: [HY000](#) ([ER_ADD_PARTITION_SUBPART_ERROR](#))
Message: Trying to Add partition(s) with wrong number of subpartitions

- Error: [1512](#) SQLSTATE: [HY000](#) ([ER_ADD_PARTITION_NO_NEW_PARTITION](#))
Message: At least one partition must be added
- Error: [1513](#) SQLSTATE: [HY000](#) ([ER_COALESCE_PARTITION_NO_PARTITION](#))
Message: At least one partition must be coalesced
- Error: [1514](#) SQLSTATE: [HY000](#) ([ER_REORG_PARTITION_NOT_EXIST](#))
Message: More partitions to reorganise than there are partitions
- Error: [1515](#) SQLSTATE: [HY000](#) ([ER_SAME_NAME_PARTITION](#))
Message: Duplicate partition name %s
- Error: [1516](#) SQLSTATE: [HY000](#) ([ER_NO_BINLOG_ERROR](#))
Message: It is not allowed to shut off binlog on this command
- Error: [1517](#) SQLSTATE: [HY000](#) ([ER_CONSECUTIVE_REORG_PARTITIONS](#))
Message: When reorganising a set of partitions they must be in consecutive order
- Error: [1518](#) SQLSTATE: [HY000](#) ([ER_REORG_OUTSIDE_RANGE](#))
Message: Reorganize of range partitions cannot change total ranges except for last partition where it can extend the range
- Error: [1519](#) SQLSTATE: [HY000](#) ([ER_PARTITION_FUNCTION_FAILURE](#))
Message: Partition function not supported in this version for this handler
- Error: [1520](#) SQLSTATE: [HY000](#) ([ER_PART_STATE_ERROR](#))
Message: Partition state cannot be defined from CREATE/ALTER TABLE
- Error: [1521](#) SQLSTATE: [HY000](#) ([ER_LIMITED_PART_RANGE](#))
Message: The %s handler only supports 32 bit integers in VALUES
- Error: [1522](#) SQLSTATE: [HY000](#) ([ER_PLUGIN_IS_NOT_LOADED](#))
Message: Plugin '%s' is not loaded
- Error: [1523](#) SQLSTATE: [HY000](#) ([ER_WRONG_VALUE](#))
Message: Incorrect %s value: '%s'
- Error: [1524](#) SQLSTATE: [HY000](#) ([ER_NO_PARTITION_FOR_GIVEN_VALUE](#))
Message: Table has no partition for value %s
- Error: [1525](#) SQLSTATE: [HY000](#) ([ER_FILEGROUP_OPTION_ONLY_ONCE](#))
Message: It is not allowed to specify %s more than once
- Error: [1526](#) SQLSTATE: [HY000](#) ([ER_CREATE_FILEGROUP_FAILED](#))
Message: Failed to create %s
- Error: [1527](#) SQLSTATE: [HY000](#) ([ER_DROP_FILEGROUP_FAILED](#))
Message: Failed to drop %s
- Error: [1528](#) SQLSTATE: [HY000](#) ([ER_TABLESPACE_AUTO_EXTEND_ERROR](#))
Message: The handler doesn't support autoextend of tablespaces

- Error: [1529](#) SQLSTATE: [HY000](#) ([ER_WRONG_SIZE_NUMBER](#))
Message: A size parameter was incorrectly specified, either number or on the form 10M
- Error: [1530](#) SQLSTATE: [HY000](#) ([ER_SIZE_OVERFLOW_ERROR](#))
Message: The size number was correct but we don't allow the digit part to be more than 2 billion
- Error: [1531](#) SQLSTATE: [HY000](#) ([ER_ALTER_FILEGROUP_FAILED](#))
Message: Failed to alter: %s
- Error: [1532](#) SQLSTATE: [HY000](#) ([ER_BINLOG_ROW_LOGGING_FAILED](#))
Message: Writing one row to the row-based binary log failed
- Error: [1533](#) SQLSTATE: [HY000](#) ([ER_BINLOG_ROW_WRONG_TABLE_DEF](#))
Message: Table definition on master and slave does not match: %s
- Error: [1534](#) SQLSTATE: [HY000](#) ([ER_BINLOG_ROW_RBR_TO_SBR](#))
Message: Slave running with --log-slave-updates must use row-based binary logging to be able to replicate row-based binary log events
- Error: [1535](#) SQLSTATE: [HY000](#) ([ER_EVENT_ALREADY_EXISTS](#))
Message: Event '%s' already exists
- Error: [1536](#) SQLSTATE: [HY000](#) ([ER_EVENT_STORE_FAILED](#))
Message: Failed to store event %s. Error code %d from storage engine.
- Error: [1537](#) SQLSTATE: [HY000](#) ([ER_EVENT_DOES_NOT_EXIST](#))
Message: Unknown event '%s'
- Error: [1538](#) SQLSTATE: [HY000](#) ([ER_EVENT_CANT_ALTER](#))
Message: Failed to alter event '%s'
- Error: [1539](#) SQLSTATE: [HY000](#) ([ER_EVENT_DROP_FAILED](#))
Message: Failed to drop %s
- Error: [1540](#) SQLSTATE: [HY000](#) ([ER_EVENT_INTERVAL_NOT_POSITIVE_OR_TOO_BIG](#))
Message: INTERVAL is either not positive or too big
- Error: [1541](#) SQLSTATE: [HY000](#) ([ER_EVENT_ENDS_BEFORE_STARTS](#))
Message: ENDS is either invalid or before STARTS
- Error: [1542](#) SQLSTATE: [HY000](#) ([ER_EVENT_EXEC_TIME_IN_THE_PAST](#))
Message: Event execution time is in the past. Event has been disabled
- Error: [1543](#) SQLSTATE: [HY000](#) ([ER_EVENT_OPEN_TABLE_FAILED](#))
Message: Failed to open mysql.event
- Error: [1544](#) SQLSTATE: [HY000](#) ([ER_EVENT_NEITHER_M_EXPR_NOR_M_AT](#))
Message: No datetime expression provided
- Error: [1545](#) SQLSTATE: [HY000](#) ([ER_COL_COUNT_DoesNT_MATCH_CORRUPTED](#))

Message: Column count of mysql.%s is wrong. Expected %d, found %d. The table is probably corrupted

- Error: [1546](#) SQLSTATE: [HY000](#) ([ER_CANNOT_LOAD_FROM_TABLE](#))

Message: Cannot load from mysql.%s. The table is probably corrupted

- Error: [1547](#) SQLSTATE: [HY000](#) ([ER_EVENT_CANNOT_DELETE](#))

Message: Failed to delete the event from mysql.event

- Error: [1548](#) SQLSTATE: [HY000](#) ([ER_EVENT_COMPILE_ERROR](#))

Message: Error during compilation of event's body

- Error: [1549](#) SQLSTATE: [HY000](#) ([ER_EVENT_SAME_NAME](#))

Message: Same old and new event name

- Error: [1550](#) SQLSTATE: [HY000](#) ([ER_EVENT_DATA_TOO_LONG](#))

Message: Data for column '%s' too long

- Error: [1551](#) SQLSTATE: [HY000](#) ([ER_DROP_INDEX_FK](#))

Message: Cannot drop index '%s': needed in a foreign key constraint

- Error: [1552](#) SQLSTATE: [HY000](#) ([ER_WARN_DEPRECATED_SYNTAX_WITH_VER](#))

Message: The syntax '%s' is deprecated and will be removed in MySQL %s. Please use %s instead

- Error: [1553](#) SQLSTATE: [HY000](#) ([ER_CANT_WRITE_LOCK_LOG_TABLE](#))

Message: You can't write-lock a log table. Only read access is possible

- Error: [1554](#) SQLSTATE: [HY000](#) ([ER_CANT_LOCK_LOG_TABLE](#))

Message: You can't use locks with log tables.

- Error: [1555](#) SQLSTATE: [23000](#) ([ER_FOREIGN_DUPLICATE_KEY](#))

Message: Upholding foreign key constraints for table '%s', entry '%s', key %d would lead to a duplicate entry

- Error: [1556](#) SQLSTATE: [HY000](#) ([ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE](#))

Message: Column count of mysql.%s is wrong. Expected %d, found %d. Created with MySQL %d, now running %d. Please use mysql_upgrade to fix this error.

- Error: [1557](#) SQLSTATE: [HY000](#) ([ER_TEMP_TABLE_PREVENTS_SWITCH_OUT_OF_RBR](#))

Message: Cannot switch out of the row-based binary log format when the session has open temporary tables

- Error: [1558](#) SQLSTATE: [HY000](#) ([ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_FORMAT](#))

Message: Cannot change the binary logging format inside a stored function or trigger

- Error: [1559](#) SQLSTATE: [HY000](#) ([ER_NDB_CANT_SWITCH_BINLOG_FORMAT](#))

Message: The NDB cluster engine does not support changing the binlog format on the fly yet

- Error: [1560](#) SQLSTATE: [HY000](#) ([ER_PARTITION_NO_TEMPORARY](#))

Message: Cannot create temporary table with partitions

- Error: [1561](#) SQLSTATE: [HY000](#) ([ER_PARTITION_CONST_DOMAIN_ERROR](#))

Message: Partition constant is out of partition function domain

- Error: [1562](#) SQLSTATE: [HY000](#) ([ER_PARTITION_FUNCTION_IS_NOT_ALLOWED](#))
Message: This partition function is not allowed
- Error: [1563](#) SQLSTATE: [HY000](#) ([ER_DDL_LOG_ERROR](#))
Message: Error in DDL log
- Error: [1564](#) SQLSTATE: [HY000](#) ([ER_NULL_IN_VALUES_LESS_THAN](#))
Message: Not allowed to use NULL value in VALUES LESS THAN
- Error: [1565](#) SQLSTATE: [HY000](#) ([ER_WRONG_PARTITION_NAME](#))
Message: Incorrect partition name
- Error: [1566](#) SQLSTATE: [25001](#) ([ER_CANT_CHANGE_TX_ISOLATION](#))
Message: Transaction isolation level can't be changed while a transaction is in progress
- Error: [1567](#) SQLSTATE: [HY000](#) ([ER_DUP_ENTRY_AUTOINCREMENT_CASE](#))
Message: ALTER TABLE causes auto_increment resequencing, resulting in duplicate entry '%s' for key '%s'
- Error: [1568](#) SQLSTATE: [HY000](#) ([ER_EVENT_MODIFY_QUEUE_ERROR](#))
Message: Internal scheduler error %d
- Error: [1569](#) SQLSTATE: [HY000](#) ([ER_EVENT_SET_VAR_ERROR](#))
Message: Error during starting/stopping of the scheduler. Error code %u
- Error: [1570](#) SQLSTATE: [HY000](#) ([ER_PARTITION_MERGE_ERROR](#))
Message: Engine cannot be used in partitioned tables
- Error: [1571](#) SQLSTATE: [HY000](#) ([ER_CANT_ACTIVATE_LOG](#))
Message: Cannot activate '%s' log
- Error: [1572](#) SQLSTATE: [HY000](#) ([ER_RBR_NOT_AVAILABLE](#))
Message: The server was not built with row-based replication
- Error: [1573](#) SQLSTATE: [HY000](#) ([ER_BASE64_DECODE_ERROR](#))
Message: Decoding of base64 string failed
- Error: [1574](#) SQLSTATE: [HY000](#) ([ER_EVENT_RECURSION_FORBIDDEN](#))
Message: Recursion of EVENT DDL statements is forbidden when body is present
- Error: [1575](#) SQLSTATE: [HY000](#) ([ER_EVENTS_DB_ERROR](#))
Message: Cannot proceed because system tables used by Event Scheduler were found damaged at server start
- Error: [1576](#) SQLSTATE: [HY000](#) ([ER_ONLY_INTEGERS_ALLOWED](#))
Message: Only integers allowed as number here
- Error: [1577](#) SQLSTATE: [HY000](#) ([ER_UNSUPPORTED_LOG_ENGINE](#))
Message: This storage engine cannot be used for log tables"
- Error: [1578](#) SQLSTATE: [HY000](#) ([ER_BAD_LOG_STATEMENT](#))
Message: You cannot '%s' a log table if logging is enabled

- Error: [1579](#) SQLSTATE: [HY000](#) ([ER_CANT_RENAME_LOG_TABLE](#))
Message: Cannot rename '%s'. When logging enabled, rename to/from log table must rename two tables: the log table to an archive table and another table back to '%s'
- Error: [1580](#) SQLSTATE: [42000](#) ([ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT](#))
Message: Incorrect parameter count in the call to native function '%s'
- Error: [1581](#) SQLSTATE: [42000](#) ([ER_WRONG_PARAMETERS_TO_NATIVE_FCT](#))
Message: Incorrect parameters in the call to native function '%s'
- Error: [1582](#) SQLSTATE: [42000](#) ([ER_WRONG_PARAMETERS_TO_STORED_FCT](#))
Message: Incorrect parameters in the call to stored function '%s'
- Error: [1583](#) SQLSTATE: [HY000](#) ([ER_NATIVE_FCT_NAME_COLLISION](#))
Message: This function '%s' has the same name as a native function
- Error: [1584](#) SQLSTATE: [23000](#) ([ER_DUP_ENTRY_WITH_KEY_NAME](#))
Message: Duplicate entry '%s' for key '%s'
- Error: [1585](#) SQLSTATE: [HY000](#) ([ER_BINLOG_PURGE_EMFILE](#))
Message: Too many files opened, please execute the command again
- Error: [1586](#) SQLSTATE: [HY000](#) ([ER_EVENT_CANNOT_CREATE_IN_THE_PAST](#))
Message: Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation.
- Error: [1587](#) SQLSTATE: [HY000](#) ([ER_EVENT_CANNOT_ALTER_IN_THE_PAST](#))
Message: Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation.
- Error: [1588](#) SQLSTATE: [HY000](#) ([ER_SLAVE_INCIDENT](#))
Message: The incident %s occurred on the master. Message: %s
- Error: [1589](#) SQLSTATE: [HY000](#) ([ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT](#))
Message: Table has no partition for some existing values
- Error: [1590](#) SQLSTATE: [HY000](#) ([ER_BINLOG_UNSAFE_STATEMENT](#))
Message: Statement is not safe to log in statement format.
- Error: [1591](#) SQLSTATE: [HY000](#) ([ER_SLAVE_FATAL_ERROR](#))
Message: Fatal error: %s
- Error: [1592](#) SQLSTATE: [HY000](#) ([ER_SLAVE_RELAY_LOG_READ_FAILURE](#))
Message: Relay log read failure: %s
- Error: [1593](#) SQLSTATE: [HY000](#) ([ER_SLAVE_RELAY_LOG_WRITE_FAILURE](#))
Message: Relay log write failure: %s
- Error: [1594](#) SQLSTATE: [HY000](#) ([ER_SLAVE_CREATE_EVENT_FAILURE](#))
Message: Failed to create %s

- Error: [1595](#) SQLSTATE: [HY000](#) ([ER_SLAVE_MASTER_COM_FAILURE](#))
Message: Master command %s failed: %s
- Error: [1596](#) SQLSTATE: [HY000](#) ([ER_BINLOG_LOGGING_IMPOSSIBLE](#))
Message: Binary logging not possible. Message: %s
- Error: [1597](#) SQLSTATE: [HY000](#) ([ER_VIEW_NO_CREATION_CTX](#))
Message: View `%s`.`%s` has no creation context
- Error: [1598](#) SQLSTATE: [HY000](#) ([ER_VIEW_INVALID_CREATION_CTX](#))
Message: Creation context of view `%s`.`%s` is invalid
- Error: [1599](#) SQLSTATE: [HY000](#) ([ER_SR_INVALID_CREATION_CTX](#))
Message: Creation context of stored routine `%s`.`%s` is invalid
- Error: [1600](#) SQLSTATE: [HY000](#) ([ER_TRG_CORRUPTED_FILE](#))
Message: Corrupted TRG file for table `%s`.`%s`
- Error: [1601](#) SQLSTATE: [HY000](#) ([ER_TRG_NO_CREATION_CTX](#))
Message: Triggers for table `%s`.`%s` have no creation context
- Error: [1602](#) SQLSTATE: [HY000](#) ([ER_TRG_INVALID_CREATION_CTX](#))
Message: Trigger creation context of table `%s`.`%s` is invalid
- Error: [1603](#) SQLSTATE: [HY000](#) ([ER_EVENT_INVALID_CREATION_CTX](#))
Message: Creation context of event `%s`.`%s` is invalid
- Error: [1604](#) SQLSTATE: [HY000](#) ([ER_TRG_CANT_OPEN_TABLE](#))
Message: Cannot open table for trigger `%s`.`%s`

B.3. Client Error Codes and Messages

Client error information comes from the following source files:

- The Error values and the symbols in parentheses correspond to definitions in the [include/errmsg.h](#) MySQL source file.
- The Message values correspond to the error messages that are listed in the [libmysql/errmsg.c](#) file. [%d](#) and [%s](#) represent numbers and strings, respectively, that are substituted into the messages when they are displayed.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: [2000](#) ([CR_UNKNOWN_ERROR](#))
Message: Unknown MySQL error
- Error: [2001](#) ([CR_SOCKET_CREATE_ERROR](#))
Message: Can't create UNIX socket (%d)
- Error: [2002](#) ([CR_CONNECTION_ERROR](#))
Message: Can't connect to local MySQL server through socket '%s' (%d)

- Error: [2003 \(CR_CONN_HOST_ERROR\)](#)
Message: Can't connect to MySQL server on '%s' (%d)
- Error: [2004 \(CR_IPSOCK_ERROR\)](#)
Message: Can't create TCP/IP socket (%d)
- Error: [2005 \(CR_UNKNOWN_HOST\)](#)
Message: Unknown MySQL server host '%s' (%d)
- Error: [2006 \(CR_SERVER_GONE_ERROR\)](#)
Message: MySQL server has gone away
- Error: [2007 \(CR_VERSION_ERROR\)](#)
Message: Protocol mismatch; server version = %d, client version = %d
- Error: [2008 \(CR_OUT_OF_MEMORY\)](#)
Message: MySQL client ran out of memory
- Error: [2009 \(CR_WRONG_HOST_INFO\)](#)
Message: Wrong host info
- Error: [2010 \(CR_LOCALHOST_CONNECTION\)](#)
Message: Localhost via UNIX socket
- Error: [2011 \(CR_TCP_CONNECTION\)](#)
Message: %s via TCP/IP
- Error: [2012 \(CR_SERVER_HANDSHAKE_ERR\)](#)
Message: Error in server handshake
- Error: [2013 \(CR_SERVER_LOST\)](#)
Message: Lost connection to MySQL server during query
- Error: [2014 \(CR_COMMANDS_OUT_OF_SYNC\)](#)
Message: Commands out of sync; you can't run this command now
- Error: [2015 \(CR_NAMEDPIPE_CONNECTION\)](#)
Message: Named pipe: %s
- Error: [2016 \(CR_NAMEDPIPEWAIT_ERROR\)](#)
Message: Can't wait for named pipe to host: %s pipe: %s (%lu)
- Error: [2017 \(CR_NAMEDPIPEOPEN_ERROR\)](#)
Message: Can't open named pipe to host: %s pipe: %s (%lu)
- Error: [2018 \(CR_NAMEDPIPESETSTATE_ERROR\)](#)
Message: Can't set state of named pipe to host: %s pipe: %s (%lu)
- Error: [2019 \(CR_CANT_READ_CHARSET\)](#)
Message: Can't initialize character set %s (path: %s)

- Error: [2020 \(CR_NET_PACKET_TOO_LARGE\)](#)
Message: Got packet bigger than 'max_allowed_packet' bytes
- Error: [2021 \(CR_EMBEDDED_CONNECTION\)](#)
Message: Embedded server
- Error: [2022 \(CR_PROBE_SLAVE_STATUS\)](#)
Message: Error on SHOW SLAVE STATUS:
- Error: [2023 \(CR_PROBE_SLAVE_HOSTS\)](#)
Message: Error on SHOW SLAVE HOSTS:
- Error: [2024 \(CR_PROBE_SLAVE_CONNECT\)](#)
Message: Error connecting to slave:
- Error: [2025 \(CR_PROBE_MASTER_CONNECT\)](#)
Message: Error connecting to master:
- Error: [2026 \(CR_SSL_CONNECTION_ERROR\)](#)
Message: SSL connection error
- Error: [2027 \(CR_MALFORMED_PACKET\)](#)
Message: Malformed packet
- Error: [2028 \(CR_WRONG_LICENSE\)](#)
Message: This client library is licensed only for use with MySQL servers having '%s' license
- Error: [2029 \(CR_NULL_POINTER\)](#)
Message: Invalid use of null pointer
- Error: [2030 \(CR_NO_PREPARE_STMT\)](#)
Message: Statement not prepared
- Error: [2031 \(CR_PARAMS_NOT_BOUND\)](#)
Message: No data supplied for parameters in prepared statement
- Error: [2032 \(CR_DATA_TRUNCATED\)](#)
Message: Data truncated
- Error: [2033 \(CR_NO_PARAMETERS_EXISTS\)](#)
Message: No parameters exist in the statement
- Error: [2034 \(CR_INVALID_PARAMETER_NO\)](#)
Message: Invalid parameter number
- Error: [2035 \(CR_INVALID_BUFFER_USE\)](#)
Message: Can't send long data for non-string/non-binary data types (parameter: %d)
- Error: [2036 \(CR_UNSUPPORTED_PARAM_TYPE\)](#)
Message: Using unsupported buffer type: %d (parameter: %d)

- Error: [2037 \(CR_SHARED_MEMORY_CONNECTION\)](#)
Message: Shared memory: %s
- Error: [2038 \(CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR\)](#)
Message: Can't open shared memory; client could not create request event (%lu)
- Error: [2039 \(CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR\)](#)
Message: Can't open shared memory; no answer event received from server (%lu)
- Error: [2040 \(CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR\)](#)
Message: Can't open shared memory; server could not allocate file mapping (%lu)
- Error: [2041 \(CR_SHARED_MEMORY_CONNECT_MAP_ERROR\)](#)
Message: Can't open shared memory; server could not get pointer to file mapping (%lu)
- Error: [2042 \(CR_SHARED_MEMORY_FILE_MAP_ERROR\)](#)
Message: Can't open shared memory; client could not allocate file mapping (%lu)
- Error: [2043 \(CR_SHARED_MEMORY_MAP_ERROR\)](#)
Message: Can't open shared memory; client could not get pointer to file mapping (%lu)
- Error: [2044 \(CR_SHARED_MEMORY_EVENT_ERROR\)](#)
Message: Can't open shared memory; client could not create %s event (%lu)
- Error: [2045 \(CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR\)](#)
Message: Can't open shared memory; no answer from server (%lu)
- Error: [2046 \(CR_SHARED_MEMORY_CONNECT_SET_ERROR\)](#)
Message: Can't open shared memory; cannot send request event to server (%lu)
- Error: [2047 \(CR_CONN_UNKNOW_PROTOCOL\)](#)
Message: Wrong or unknown protocol
- Error: [2048 \(CR_INVALID_CONN_HANDLE\)](#)
Message: Invalid connection handle
- Error: [2049 \(CR_SECURE_AUTH\)](#)
Message: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)
- Error: [2050 \(CR_FETCH_CANCELED\)](#)
Message: Row retrieval was canceled by mysql_stmt_close() call
- Error: [2051 \(CR_NO_DATA\)](#)
Message: Attempt to read column without prior row fetch
- Error: [2052 \(CR_NO_STMT_METADATA\)](#)
Message: Prepared statement contains no metadata
- Error: [2053 \(CR_NO_RESULT_SET\)](#)
Message: Attempt to read a row while there is no result set associated with the statement

- Error: [2054 \(CR_NOT_IMPLEMENTED\)](#)
Message: This feature is not implemented yet
- Error: [2055 \(CR_SERVER_LOST_EXTENDED\)](#)
Message: Lost connection to MySQL server at '%s', system error: %d
- Error: [2056 \(CR_STMT_CLOSED\)](#)
Message: Statement closed indirectly because of a preceeding %s() call

Appendix C. MySQL Change History

This appendix lists the changes from version to version in the MySQL source code through the latest version of MySQL 5.1, which is currently MySQL 5.1.21-beta. Starting with MySQL 5.0, we began offering a new version of the Manual for each new series of MySQL releases (5.0, 5.1, and so on). For information about changes in previous release series of the MySQL database software, see the corresponding version of this Manual. For information about legacy versions of the MySQL software through the 4.1 series, see *MySQL 3.23, 4.0, 4.1 Reference Manual*.

We update this section as we add new features in the 5.1 series, so that everybody can follow the development process.

Note that we tend to update the manual at the same time we make changes to MySQL. If you find a recent version of MySQL listed here that you can't find on our download page (<http://dev.mysql.com/downloads/>), it means that the version has not yet been released.

The date mentioned with a release version is the date of the last BitKeeper ChangeSet on which the release was based, not the date when the packages were made available. The binaries are usually made available a few days after the date of the tagged ChangeSet, because building and testing all packages takes some time.

The manual included in the source and binary distributions may not be fully accurate when it comes to the release changelog entries, because the integration of the manual happens at build time. For the most up-to-date release changelog, please refer to the online version instead.

C.1. Changes in release 5.1.x (Development)

An overview of which features were added in MySQL 5.1 can be found here: [Section 1.6.1, “What's New in MySQL 5.1”](#).

For a full list of changes, please refer to the changelog sections for each individual 5.1.x release.

C.1.1. Changes in release 5.1.23 (Not yet released)

This is a new Beta development release, fixing recently discovered bugs.

Note

This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

Functionality added or changed:

- If a **MyISAM** table is created with no **DATA DIRECTORY** option, the **.MYD** file is created in the database directory. By default, if **MyISAM** finds an existing **.MYD** file in this case, it overwrites it. The same applies to **.MYI** files for tables created with no **INDEX DIRECTORY** option. To suppress this behavior, start the server with the new **--keep_files_on_create** option, in which case **MyISAM** will not overwrite existing files and returns an error instead. ([Bug#29325](#))

Bugs fixed:

- **STR_TO_DATE()** displayed an error message that referred to **STR_TO_TIME()**. ([Bug#27014](#))
- **mysql_upgrade** could run binaries dynamically linked against incorrect versions of shared libraries. ([Bug#28560](#))
- MySQL failed to generate or retrieve an **AUTO_INCREMENT** primary key for **InnoDB** tables with user-defined partitioning. ([Bug#27405](#))
- The query cache does not support retrieval of statements for which column level access control applies, but the server was still caching such statements, thus wasting memory. ([Bug#30269](#))
- Using **HANDLER** to open a table having a storage engine not supported by **HANDLER** properly returned an error, but also improperly prevented the table from being dropped by other connections. ([Bug#25856](#))

- Statements within stored procedures ignored the value of the `low_priority_updates` system variable. (Bug#28570)
- When `mysqlslap` was given a query to execute from a file via a `--query=file_name` option, it executed the query one too many times. (Bug#29803)
- A `SELECT` with more than 31 nested dependent subqueries returned an incorrect result. (Bug#27352)
- Some character mappings in the `ascii.xml` file were incorrect. (Bug#27562)
- Some `SHOW` statements and `INFORMATION_SCHEMA` queries could expose information not allowed by the user's access privileges. (Bug#27629)
- If a `LIMIT` clause was present, the server could fail to consider indexes that could be used for `ORDER BY` or `GROUP BY`. (Bug#28404)
- `mysqlbinlog` produced incorrectly formatted `DATETIME` and `TIMESTAMP` values. (Bug#27894)
- `GROUP BY` on `BIT` columns produced incorrect results. (Bug#30219)

C.1.2. Changes in release 5.1.22 (Not yet released)

This is a new Beta development release, fixing recently discovered bugs.

Note

This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

Functionality added or changed:

- There is a new `innodb_autoinc_lock_mode` system variable to configure the locking behavior that InnoDB uses for generating auto-increment values. The default behavior now is slightly different from before, which involves a minor incompatibility for multiple-row inserts that specify an explicit value for the auto-increment column in some but not all rows. See [Section 13.5.6.3, “How AUTO_INCREMENT Handling Works in InnoDB”](#).

Bugs fixed:

- **NDB Cluster**: Backups of `TIMESTAMP` columns made with `ndb_restore` on a MySQL Cluster using data nodes hosts of one endian could not be used to restore the cluster's data to data node hosts of the other endian. (Bug#30134)
- **NDB Cluster** (Replication): Multi-master replication setups did not handle `--log-slave-updates` correctly. (Bug#30017)
- `mysql_install_db` could fail to find its message file. (Bug#30678)
- Non-range queries of the form `SELECT ... FROM ... WHERE keypart_1=const, ..., keypart_n=const ORDER BY ... FOR UPDATE` sometimes were unnecessarily blocked waiting for a lock if another transaction was using `SELECT ... FOR UPDATE` on the same table. (Bug#29804)
- Under some circumstances, a UDF initialization function could be passed incorrect argument lengths. (Bug#29804)
- `CONNECTION_ID()` always returned 0 for the embedded server (`libmysqld`). (Bug#30389)
- The `mysql_list_fields()` C API function incorrectly set `MYSQL_FIELD::decimals` for some view columns. (Bug#29306)
- Read lock requests that were blocked by a pending write lock request were not allowed to proceed if the statement requesting the write lock was killed. (Bug#21281)
- Memory corruption occurred for some queries with a top-level `OR` operation in the `WHERE` condition if they contained equality pre-

dicates and other sargable predicates in disjunctive parts of the condition. ([Bug#30396](#))

- The server created temporary tables for filesort operations in the working directory, not in the directory specified by the `tmpdir` system variable. ([Bug#30287](#))
- Using `KILL QUERY` or `KILL CONNECTION` to kill a `SELECT` statement caused a server crash if the query cache was enabled. ([Bug#30201](#))
- Operations that used the time zone replicated the time zone only for successful operations, but did not replicate the time zone for errors that need to know it. ([Bug#29536](#))
- `mysqldump` from the MySQL 5.1.21 distribution could not be used to create a dump from a MySQL 5.1.20 or older server. ([Bug#30123](#))
- When using a combination of `HANDLER . . . READ` and `DELETE` on a table, MySQL continued to open new copies of the table every time, leading to an exhaustion of file descriptors. This was caused in MySQL 5.1.15 by a fix for [Bug#21587](#); the current fix consists of reverting the earlier fix. ([Bug#29474](#))
- Tables using the `InnoDB` storage engine incremented `AUTO_INCREMENT` values incorrectly with `ON DUPLICATE KEY UPDATE`. ([Bug#28781](#))

C.1.3. Changes in MySQL 5.1.20 Carrier Grade Edition

This section contains change history information for MySQL Cluster 5.1 Carrier Grade Edition releases based on MySQL 5.1.22.

C.1.3.1. Changes in release MySQL 5.1.22-ndb-6.3.2-beta (Not yet released)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.19-ndb-6.3.0 and MySQL 5.1.19-ndb-6.3.1, as well as all bugfixes and feature changes which were added in the mainline 5.1.21 and 5.1.22 releases; information about these can be found in [Section C.1.4, “Changes in release 5.1.21 \(16 August 2007\)”](#) and [Section C.1.2, “Changes in release 5.1.22 \(Not yet released\)”](#).

Functionality added or changed:

- It is now possible to control whether fixed-width or variable-width storage is used for a given column of an `NDB` table by means of the `COLUMN_FORMAT` specifier as part of the column's definition in a `CREATE TABLE` or `ALTER TABLE` statement.

It is also possible to control whether a given column of an `NDB` table is stored in memory or on disk, using the `STORAGE` specifier as part of the column's definition in a `CREATE TABLE` or `ALTER TABLE` statement.

For permitted values and other information about `COLUMN_FORMAT` and `STORAGE`, see [Section 12.1.10, “CREATE TABLE Syntax”](#).
- A new cluster management server startup option `--bind-address` makes it possible to restrict management client connections to `ndb_mgmd` to a single host and port. For more information, see [Section 16.6.5.3, “Command Options for ndb_mgmd”](#).
- The protocol for handling global checkpoints has been changed. It is now possible to control how often the GCI number is updated, and how often global checkpoints are written to disk, using the `TimeBetweenEpochs` configuration parameter. This improves the reliability and performance of MySQL Cluster Replication.

GCPs handled using the new protocol are sometimes referred to as “micro-GCPs”.

For more information, see [TimeBetweenEpochs](#).

This release fixes the following bugs:

- For micro-GCPs, fixed the assignment of “fake” CGI events so that they do not cause buckets to be sent out of order. Now, when assigning a GCI to a non-GCI event (that is, creating a pseudo-GCI or “fake” CGI), the GCI that is to arrive is always initiated, even if no known GCI exists, which could occur in the event of a node failure. (Bug#30884)

C.1.3.2. Changes in release MySQL 5.1.22-ndb-6.2.6-beta (Not yet released)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.18-ndb-6.2.1, MySQL 5.1.18-ndb-6.2.2 (see [Section C.1.9, “Changes in MySQL 5.1.18 Carrier Grade Edition”](#)), MySQL 5.1.19-ndb-6.2.3, MySQL 5.1.19-ndb-6.2.4, and MySQL 5.1.22-ndb-6.2.5, as well as all bugfixes and feature changes which were added in the mainline 5.1.20, 5.1.21 and 5.1.22 releases; information about these can be found in [Section C.1.5, “Changes in release 5.1.20 \(25 June 2007\)”](#), [Section C.1.4, “Changes in release 5.1.21 \(16 August 2007\)”](#), and [Section C.1.2, “Changes in release 5.1.22 \(Not yet released\)”](#). Also included are most (but not all) bugfixes made in the MCCGE 6.1.x series through MySQL 5.1.15-ndb-6.1.16.

This release fixes the following bugs:

- For micro-GCPs, fixed the assignment of “fake” CGI events so that they do not cause buckets to be sent out of order. Now, when assigning a GCI to a non-GCI event (that is, creating a pseudo-GCI or “fake” CGI), the GCI that is to arrive is always initiated, even if no known GCI exists, which could occur in the event of a node failure. (Bug#30884)
- [EXPLAIN PARTITIONS](#) reported partition usage by queries on [NDB](#) tables according to the standard MySQL hash function than the hash function used in the [NDB](#) storage engine. (Bug#29550)

C.1.3.3. Changes in release MySQL 5.1.22-ndb-6.2.5-beta (06 September 2007)

This is a new Beta development release, fixing recently discovered bugs.

Like all releases for MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.22-ndb-6.2.5. The file `mysqlcom-5.1.22-ndb-6.2.5-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.18-ndb-6.2.1, MySQL 5.1.18-ndb-6.2.2 (see [Section C.1.9, “Changes in MySQL 5.1.18 Carrier Grade Edition”](#)), MySQL 5.1.19-ndb-6.2.3, and MySQL 5.1.19-ndb-6.2.4, as well as all bugfixes and feature changes which were added in the mainline 5.1.20, 5.1.21 and 5.1.22 releases; information about these can be found in [Section C.1.5, “Changes in release 5.1.20 \(25 June 2007\)”](#), [Section C.1.4, “Changes in release 5.1.21 \(16 August 2007\)”](#), and [Section C.1.2, “Changes in release 5.1.22 \(Not yet released\)”](#). Also included are most (but not all) bugfixes made in the MCCGE 6.1.x series through MySQL 5.1.15-ndb-6.1.16.

Functionality added or changed:

- It is now possible to control whether fixed-width or variable-width storage is used for a given column of an [NDB](#) table by means of the [COLUMN_FORMAT](#) specifier as part of the column's definition in a [CREATE TABLE](#) or [ALTER TABLE](#) statement.

It is also possible to control whether a given column of an [NDB](#) table is stored in memory or on disk, using the [STORAGE](#) specifier as part of the column's definition in a [CREATE TABLE](#) or [ALTER TABLE](#) statement.

For permitted values and other information about [COLUMN_FORMAT](#) and [STORAGE](#), see [Section 12.1.10, “CREATE TABLE Syntax”](#).

- A new cluster management server startup option `--bind-address` makes it possible to restrict management client connections to

`ndb_mgmd` to a single host and port. For more information, see [Section 16.6.5.3, “Command Options for ndb_mgmd”](#).

- The following improvements have been made in the `ndb_size.pl` utility:
 - The script is now able to be used with multiple databases; lists of databases and tables can be excluded from analysis.
 - Schema name information has been added to index table calculations.
 - The database name is now an optional parameter, the exclusion of which causes all databases to be examined.
 - If selecting from `INFORMATION_SCHEMA` fails, the script now attempts to fall back to `SHOW TABLES`.
 - A `--real_table_name` option has been added; this designates a table to handle unique index size calculations.
 - The report title has been amended to cover cases where more than one database is being analyzed. ([Bug#28683](#))

Support for a `--socket` option was also added. ([Bug#28253](#))

For more information, see [Section 16.9.15, “ndb_size.pl — NDBCluster Size Requirement Estimator”](#).

- The protocol for handling global checkpoints has been changed. It is now possible to control how often the GCI number is updated, and how often global checkpoints are written to disk, using the `TimeBetweenEpochs` configuration parameter. This improves the reliability and performance of MySQL Cluster Replication.

GCPs handled using the new protocol are sometimes referred to as “micro-GCPs”.

For more information, see [TimeBetweenEpochs](#).

This release fixes the following bugs:

- An attempt to perform a `SELECT ... FROM INFORMATION_SCHEMA.TABLES` whose result included information about NDB tables for which the user had no privileges could crash the MySQL Server on which the query was performed. ([Bug#26793](#))
- `ndb_size.pl` failed on tables with `FLOAT` columns whose definitions included commas (for example, `FLOAT(6,2)`). ([Bug#29228](#))
- `DELETE FROM table WHERE primary_key IN (value_list)`, where the `value_list` contained more than one value, called from an `AFTER DELETE` trigger on an NDB table, caused `mysqld` to crash. ([Bug#30337](#))
- A query using joins between several large tables and requiring unique index lookups failed to complete, eventually returning `UNKNOWN ERROR` after a very long period of time. This occurred due to inadequate handling of instances where the Transaction Coordinator ran out of `TransactionBufferMemory`, when the cluster should have returned NDB error code 4012 (`REQUEST NDBD TIME-OUT`). ([Bug#28804](#))
- **Cluster Replication:** Cluster replication did not handle large `VARCHAR` columns correctly. ([Bug#29904](#))
- When handling `BLOB` columns, the addition of read locks to the lock queue was not handled correctly. ([Bug#30764](#))
- **Disk Data:** `DROP TABLESPACE` failed when a table that had been using the tablespace contained a `BLOB` column. ([Bug#29102](#))
- **APIs:** A call to `CHECK_TIMEDOUT_RET()` in `mgmapi.cpp` should have a call to `DEBUG_CHECK_TIMEDOUT_RET()`. ([Bug#30681](#))
- Discovery of NDB tables did not work correctly with `INFORMATION_SCHEMA`. ([Bug#30667](#))
- A filesystem close could fail during a node or system restart. ([Bug#30646](#))
- Using the `--ndb-cluster-connection-pool` option could cause DDL statements to be executed twice. ([Bug#30598](#))

C.1.4. Changes in release 5.1.21 (16 August 2007)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

Note

Subsequent to release, it was discovered that on some platforms, `mysql_install_db` could fail to find its message file, resulting in error messages of the following form:

```
shell> mysql_install_db
Installing MySQL system tables...
070830  9:33:24 [ERROR] Can't find messagefile 'path/share/english/errmsg.sys'
070830  9:33:24 [ERROR] Aborting
```

To deal with this problem, specify a `--language` option to specify the proper pathname to the language file directory. For example:

```
shell> mysql_install_db --language=/path/to/share/english/
```

This problem is corrected in MySQL 5.1.22.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- **Incompatible change:** In MySQL 5.1.6, when log tables were implemented, the default log destination for the general query and slow query log was `TABLE`. This default has been changed to `FILE`, which is compatible with MySQL 5.0, but incompatible with earlier releases of MySQL 5.1 from 5.1.6 to 5.1.20. If you are upgrading from MySQL 5.0 to this release, no logging option changes should be necessary. However, if you are upgrading from 5.1.6 through 5.1.20 to this release and were using `TABLE` logging, use the `--log-output=TABLE` option explicitly to preserve your server's table-logging behavior. (Bug#29993)
- The `--syslog` option that was introduced in 5.1.20 for `mysqld_safe` (to send error output to `syslog`) did not work correctly: Error output was buffered and not logged immediately. This has been corrected. In addition, some feature changes were made:

Important

The default `mysqld_safe` logging behavior now is `--skip-syslog` rather than `--syslog`, which is compatible with the default behavior of writing an error log file for releases prior to 5.1.20.

- A new option, `--syslog-tag=tag`, modifies the default tags written by `mysqld_safe` and `mysqld` to `syslog` to be `mysqld_safe-tag` and `mysqld-tag` rather than the default tags of `mysqld_safe` and `mysqld`. (Bug#29992)
- Several programs now accept `--debug-check` and `--debug-info` options: `mysql`, `mysqladmin`, `mysqlbinlog`, `mysqlcheck`, `mysqldump`, `mysqlimport`, `mysqlshow`, `mysqlslap`, `mysqldtest`, `mysql_upgrade`. (Note: `mysql` and `mysqldtest` already accepted `--debug-info`.) `--debug-check` prints debugging information at program exit. `--debug-info` is similar but also prints memory and CPU usage statistics. This patch also corrects a problem for `mysql` that `--debug-info` did not display statistics at exit time. (Bug#30127)
- Transaction support in the `FEDERATED` storage engine has been disabled due to issues with multiple active transactions and sessions on the same `FEDERATED` table. (Bug#29875)
- Previously, prepared statements processed using `PREPARE` and `EXECUTE` were not subject to caching in the query cache if they contained any `?` parameter markers. This limitation has been lifted. (Bug#29318)
- Replication between master and slaves now supports different column numbers within a table on both master and slave. The rules for replication where the table definitions are different has also changed. This supercedes the functionality for replication from the master table to a slave table with more columns that was added in MySQL 5.1.12. For more information, see [Section 15.3.1.20](#),

“Replication with Fewer Columns on the Slave”.

- The SQL thread on a slave now is always allowed to enter [InnoDB](#) even if this would exceed the limit imposed by the [innodb_thread_concurrency](#) system variable. In cases of high load on the slave server (when [innodb_thread_concurrency](#) is reached), this change helps the slave stay more up to date with the master; in the previous behavior, the SQL thread was competing for resources with all client threads active on the slave server. ([Bug#25078](#))

Bugs fixed:

- **Incompatible change:** Several issues were identified for stored programs (stored functions and procedures, triggers, and events) and views containing non-ASCII symbols. These issues involved conversion errors due to incomplete character set information when translating these objects to and from stored format, such as:
 - Parsing the original object definition so that it can be stored.
 - Compiling the stored definition into executable form when the object is invoked.
 - Retrieval of object definitions from [INFORMATION_SCHEMA](#) tables.
 - Displaying the object definition in [SHOW](#) statements. This issue also affected [mysqldump](#), which uses [SHOW](#).

The fix for the problems is to store character set information from the object creation context so that this information is available when the object needs to be used later. The context includes the client character set, the connection character set and collation, and the collation of the database with which the object is associated.

As a result of the patch, several tables have new columns:

- In the [mysql](#) database, the [proc](#) and [event](#) tables now have these columns: [character_set_client](#), [collation_connection](#), [db_collation](#), [body_utf8](#).
- In [INFORMATION_SCHEMA](#), the [VIEWS](#) table now has these columns: [CHARACTER_SET_CLIENT](#), [COLLATION_CONNECTION](#). The [ROUTINES](#), [TRIGGERS](#), and [EVENT](#) tables now have these columns: [CHARACTER_SET_CLIENT](#), [COLLATION_CONNECTION](#), [DATABASE_COLLATION](#).

These columns store the session values of the [character_set_client](#) and [collation_connection](#) system variables, and the collation of the database with which the object is associated. The values are those in effect at object creation time. (The saved database collation is not the value of the [collation_database](#) system variable, which applies to the default database; the database that contains the object is not necessarily the default database.)

Several [SHOW](#) statements now display additional columns corresponding to the new table columns. These statements are: [SHOW CREATE EVENT](#), [SHOW CREATE FUNCTION](#), [SHOW CREATE PROCEDURE](#), [SHOW CREATE VIEW](#), [SHOW EVENTS](#), [SHOW FUNCTION STATUS](#), [SHOW PROCEDURE STATUS](#), [SHOW TRIGGERS](#).

A new statement, [SHOW CREATE TRIGGER](#) is introduced and is used by [mysqldump](#) for producing [CREATE TRIGGER](#) statements.

([Bug#11986](#), [Bug#16291](#), [Bug#19443](#), [Bug#21249](#), [Bug#25212](#), [Bug#25221](#), [Bug#30027](#))

Subsequent to the patch just described, it was discovered that the patch broke [mysql_upgrade](#); this has been corrected. ([Bug#30029](#))

Important

To avoid warnings from the server about the use of old definitions from any release prior to 5.1.21, you should dump stored programs and views with [mysqldump](#) after upgrading to 5.1.21, and then reload them to recreate them with new definitions. Invoke [mysqldump](#) with a [--default-character-set](#) option that names the non-ASCII character set that was used for the definitions when the objects were originally defined.

- Storage engine error conditions in row-based replication were not correctly reported to the user. ([Bug#29570](#))
- **NDB Cluster:** A problem with the fix for [Bug#29354](#) caused an assertion when two local checkpoints were run during node recovery.
- **NDB Cluster:** The server would not compile with **NDB** support on AIX 5.2. ([Bug#10776](#))

- **NDB Cluster** (Replication): Inconsistencies could occur between the master and the slave when replicating Disk Data tables. (Bug#19259)
- **NDB Cluster** (Replication): `mysqld` would segfault on startup when the **NDB** storage engine was enabled and the default character set was a strictly multi-byte character set such as UCS2. (Bug#27404)

This issue does *not* apply to character sets that can contain single-byte characters in addition to multi-byte characters such as UTF-8.

Additional issues remain with regard to the use of multi-byte character sets in MySQL Cluster Replication; see [Section 16.10.3, “Known Issues in MySQL Cluster Replication”](#), for more information.
- **NDB Cluster**: Warnings and errors generated by `ndb_config --config-file=file` were sent to `stdout`, rather than to `stderr`. (Bug#25941)
- **NDB Cluster** (Disk Data): Performing Disk Data schema operations during a node restart could cause forced shutdowns of other data nodes. (Bug#29501)
- **NDB Cluster** (Disk Data): Disk data meta-information that existed in `ndb` might not be visible to `mysqld`. (Bug#28720)
- **NDB Cluster** (Disk Data): The number of free extents was incorrectly reported for some tablespaces. (Bug#28642)
- **NDB Cluster** (Cluster Replication): When executing a statement where `binlog_format=statement`, the result of the statement was logged both as a statement and as rows. (Bug#29222)
- **NDB Cluster**: Replica redo logs were inconsistently handled during a system restart. (Bug#29354)
- **NDB Cluster**: When restarting a data node, queries could hang during that node's start phase 5, and only continue once the node entered phase 6. (Bug#29364)
- **NDB Cluster**: When a node failed to respond to a `COPY_GCI` signal as part of a global checkpoint, the master node was killed instead instead of the node that actually failed. (Bug#29331)
- **NDB Cluster**: An invalid comparison made during `REDO` validation that could lead to an `ERROR WHILE READING REDO LOG` condition. (Bug#29118)
- **NDB Cluster**: The wrong data pages were sometimes invalidated following a global checkpoint. (Bug#29067)
- **NDB Cluster**: If at least 2 files were involved in `REDO` invalidation, then file 0 of page 0 was not updated and so pointed to an invalid part of the redo log. (Bug#29057)
- The server acquired a global mutex for temporary tables, although such tables are thread-specific. This affected performance by blocking other threads. (Bug#27062)
- `INSERT DELAYED` statements on a master server are replicated as non-`DELAYED` inserts on slaves (which is normal, to preserve serialization), but the inserts on the slave did not use concurrent inserts. Now `INSERT DELAYED` on a slave is converted to a concurrent insert when possible, and to a normal insert otherwise. (Bug#29152)
- If one thread was performing concurrent inserts, other threads reading from the same table using equality key searches could see the index values for new rows before the data values had been written, leading to reports of table corruption. (Bug#29838)
- On Windows, client libraries lacked symbols required for linking. (Bug#30118)
- On Windows, the CMake build process did not produce the embedded server library or related binaries. (Bug#29903)
- `SESSION_USER()` returned garbage data (rather than the correct value of the empty string) when executed by a slave SQL thread. (Bug#29878)
- For the `SHOW TABLE TYPES` statement, the server sent incorrect output to clients, possibly causing them to crash. (Bug#30036)
- Coercion of ASCII values to character sets that are a superset of ASCII sometimes was not done, resulting in `ILLEGAL MIX OF COLLATIONS` errors. These cases now are resolved using repertoire, a new string expression attribute (see [Section 9.6, “String Repertoire”](#)). (Bug#28875)
- `REPAIR TABLE ... USE_FRM` could corrupt tables. (Bug#29980)
- `FEDERATED` tables had an artificially low maximum of key length. (Bug#26909)

- In some cases, `INSERT INTO ... SELECT ... GROUP BY` could insert rows even if the `SELECT` by itself produced an empty result. (Bug#29717)
- In a stored function or trigger, when `InnoDB` detected deadlock, it attempted rollback and displayed an incorrect error message (`EXPLICIT OR IMPLICIT COMMIT IS NOT ALLOWED IN STORED FUNCTION OR TRIGGER`). Now `InnoDB` returns an error under these conditions and does not attempt rollback. Rollback is handled outside of `InnoDB` above the function/trigger level. (Bug#24989)
- `--myisam-recover=""` (empty option value) did not disable `MyISAM` recovery. (Bug#29856)
- Very long prepared statements in stored procedures could cause a server crash. (Bug#29856)
- Index creation could fail due to truncation of key values to the maximum key length rather than to a multiple of the maximum character length. (Bug#28125)
- `mysql_setpermission` tried to grant global-only privileges at the database level. (Bug#14618)
- An error that happened inside `INSERT`, `UPDATE`, or `DELETE` statements performed from within a stored function or trigger could cause inconsistency between master and slave servers. (Bug#27417)
- An assertion failure occurred within `yaSSL` for very long keys. (Bug#29784)
- Repeatedly accessing a view in a stored procedure (for example, in a loop) caused a small amount of memory to be allocated per access. Although this memory is deallocated on disconnect, it could be a problem for a long running stored procedures that make repeated access of views. (Bug#29834)
- The `IS_UPDATABLE` column in the `INFORMATION_SCHEMA.VIEWS` table was not always set correctly. (Bug#30020)
- A slave running with `--log-slave-updates` would fail to write `INSERT DELAY IGNORE` statements to its binary log, resulting in different binary log contents on the master and slave. (Bug#29571)
- `InnoDB` could crash if the server was shut down while `innodb_table_monitor` was running. (Bug#28254)
- If `MySQL/InnoDB` crashed very quickly after starting up, it would not force a checkpoint. In this case, `InnoDB` would skip crash recovery at next startup, and the database would become corrupt. Fix: If the redo log scan at `InnoDB` startup goes past the last checkpoint, force crash recovery. (Bug#23710)
- A maximum of 4TB `InnoDB` free space was reported by `SHOW TABLE STATUS`, which is incorrect on systems with more than 4TB space. (Bug#29097)
- `InnoDB` refused to start on some versions of FreeBSD with `LinuxThreads`. This is fixed by enabling file locking on FreeBSD. (Bug#29155)
- Certain statements with unions, subqueries, and joins could result in huge memory consumption. (Bug#29582)
- `SHOW` statements were being written to the slow query log that should not have been. (Bug#30000)
- Use of local variables with non-ASCII names in stored procedures crashed the server. (Bug#30120)
- `INSERT ... VALUES(CONNECTION_ID(), ...)` statements were written to the binary log in such a way that they could not be properly restored. (Bug#29928)
- Prepared statements containing `CONNECTION_ID()` could be written improperly to the binary log. (Bug#30200)
- `mysql_install_db` could fail to find script files that it needs. (Bug#28585)
- On Windows, executables did not include Vista manifests. (Bug#24732)
- On Windows, the server used 10MB of memory for each connection thread, resulting in memory exhaustion. Now each thread uses 1MB. (Bug#20815)
- Dropping a temporary `InnoDB` table that had been locked with `LOCK TABLES` caused a server crash. (Bug#24918)
- Log table locking was redesigned, eliminating several lock-related problems:
 - Truncating `mysql.slow_log` in a stored procedure after use of a cursor caused the thread to lock. (Bug#17876)

- Flushing a log table resulted in warnings. ([Bug#23044](#))
- The server would hang when performing concurrent `ALTER TABLE` or `TRUNCATE TABLE` statements against the log tables. ([Bug#25422](#))
- Changing the value of the `general_log` system variable while a global read lock was in place resulted in deadlock. ([Bug#29129](#))

The changes provide more well-defined interface characteristics. See [Section 5.11.1, “Selecting General Query and Slow Query Log Output Destinations”](#).

- LOCK TABLES did not pre-lock tables used in triggers of the locked tables. Unexpected locking behavior and statement failures similar to `FAILED: 1100: TABLE 'xx' WAS NOT LOCKED WITH LOCK TABLES` could result. ([Bug#29929](#))
- Creating an event to be executed at a time close to the end of the allowed range (2038-01-19 03:14:07 UTC) would cause the server to crash. ([Bug#28641](#))
- On Windows, Instance Manager would crash if an instance object failed to initialize during startup. This could happen if an incorrect `mysqld` path was supplied in the configuration file. ([Bug#28012](#))
- Instance Manager had a race condition when it received a shutdown request while a guarded `mysqld` instance was starting such that it could fail to stop the `mysqld` instance. ([Bug#28030](#))
- Fast `ALTER TABLE` (that works without rebuilding the table) acquired duplicate locks in the storage engine. In `MyISAM`, if `ALTER TABLE` was issued under `LOCK TABLE`, it caused all data inserted after `LOCK TABLE` to disappear. ([Bug#28838](#))
- After the first read of a `TEMPORARY` table, `CHECK TABLE` could report the table as being corrupt. ([Bug#26325](#))
- The server was blocked from opening other tables while the `FEDERATED` engine was attempting to open a remote table. Now the server does not check the correctness of a `FEDERATED` table at `CREATE TABLE` time, but waits until the table actually is accessed. ([Bug#25679](#))
- On Mac OS X, shared-library installation pathnames were incorrect. ([Bug#28544](#))
- For `MyISAM` tables on Windows, `INSERT`, `DELETE`, or `UPDATE` followed by `ALTER TABLE` within `LOCK TABLES` could cause table corruption. ([Bug#29957](#))
- When determining which transaction to kill after deadlock has been detected, `InnoDB` now adds the number of locks to a transaction's weight, and avoids killing transactions that have modified non-transactional tables. This should reduce the likelihood of killing long-running transactions containing `SELECT ... FOR UPDATE` or `INSERT/REPLACE INTO ... SELECT` statements, and of causing partial updates if the target is a `MyISAM` table. ([Bug#21293](#))
- When using a `FEDERATED` table, the value of `last_insert_id()` would not correctly update the C API interface, which would affect the autogenerated ID returned both through the C API and the MySQL protocol, affecting Connectors that used the protocol and/or C API. ([Bug#25714](#))
- Optimization of queries with `DETERMINISTIC` stored functions in the `WHERE` clause was ineffective: A sequential scan was always used. ([Bug#29338](#))
- `SQL_BIG_RESULT` had no effect for `CREATE TABLE ... SELECT SQL_BIG_RESULT ...` statements. ([Bug#15130](#))
- For `InnoDB` tables, MySQL unnecessarily sorted records in certain cases when the records were retrieved by `InnoDB` in the proper order already. ([Bug#28591](#))
- `EXPLAIN` produced `Impossible where` for statements of the form `SELECT ... FROM t WHERE c=0`, where `c` was an `ENUM` column defined as a primary key. ([Bug#29661](#))
- On Windows, `ALTER TABLE` hung if records were locked in share mode by a long-running transaction. ([Bug#29644](#))
- A field packet with `NULL` fields caused a `libmysqlclient` crash. ([Bug#29494](#))
- A byte-order issue in writing a spatial index to disk caused bad index files on some systems. ([Bug#29070](#))
- `mysqldump` produced output that incorrectly discarded the `NO_AUTO_VALUE_ON_ZERO` value of the `SQL_MODE` variable after dumping triggers. ([Bug#29788](#))

- Adding `DISTINCT` could cause incorrect rows to appear in a query result. ([Bug#29911](#))
- Killing an `INSERT DELAYED` thread caused a server crash. ([Bug#29431](#))
- For updates to `InnoDB` tables, a `TIMESTAMP` column with the `ON UPDATE CURRENT_TIMESTAMP` attribute could be updated even when no values actually changed. ([Bug#29310](#))
- The special “zero” `ENUM` value was coerced to the normal empty string `ENUM` value during a column-to-column copy. This affected `CREATE ... SELECT` statements and `SELECT` statements with aggregate functions on `ENUM` columns in the `GROUP BY` clause. ([Bug#29360](#))
- Conversion of ASCII DEL (`0x7F`) to Unicode incorrectly resulted in QUESTION MARK (`0x3F`) rather than DEL. ([Bug#29499](#))
- A left join between two views could produce incorrect results. ([Bug#29604](#))
- For `MEMORY` tables, the `index_merge` union access method could return incorrect results. ([Bug#29740](#))
- If query execution involved a temporary table, `GROUP_CONCAT()` could return a result with an incorrect character set. ([Bug#29850](#))
- Slave servers could incorrectly interpret an out-of-memory error from the master and reconnect using the wrong binary log position. ([Bug#24192](#))
- Comparison of `TIME` values using the `BETWEEN` operator led to string comparison, producing incorrect results in some cases. Now the values are compared as integers. ([Bug#29739](#))
- An incorrect result was returned when comparing string values that were converted to `TIME` values with `CAST()`. ([Bug#29555](#))
- If `InnoDB` reached its limit on the number of concurrent transactions (1023), it wrote a descriptive message to the error log but returned a misleading error message to the client, or an assertion failure occurred. ([Bug#18828](#))
- On Windows, the `mysql` client died if the user entered a statement and Return after entering Control-C. ([Bug#29469](#))
- For the general query log, logging of prepared statements executed via the C API differed from logging of prepared statements performed with `PREPARE` and `EXECUTE`. Logging for the latter was missing the `Prepare` and `Execute` lines. ([Bug#13326](#))
- Under heavy load with a large query cache, invalidating part of the cache could cause the server to freeze (that is, to be unable to service other operations until the invalidation was complete). ([Bug#21074](#))
- If an operation had an `InnoDB` table, and two triggers, `AFTER UPDATE` and `AFTER INSERT`, competing for different resources (such as two distinct `MyISAM` tables), the triggers were unable to execute concurrently. In addition, `INSERT` and `UPDATE` statements for the `InnoDB` table were unable to run concurrently. ([Bug#26141](#))
- On 64-bit platforms, the filesort code (for queries with `GROUP BY` or `ORDER BY`) could crash due to an incorrect pointer size. ([Bug#29610](#))
- Using the `DATE()` function in a `WHERE` clause did not return any records after encountering `NULL`. However, using `TRIM` or `CAST` produced the correct results. ([Bug#29898](#))
- Using the `--skip-add-drop-table` option with `mysqldump` generated incorrect SQL if the database included any views. The recreation of views requires the creation and removal of temporary tables. This option suppressed the removal of those temporary tables. The same applied to `--compact` since this option also invokes `--skip-add-drop-table`. ([Bug#28524](#))
- A race condition in the interaction between `MyISAM` and the query cache code caused the query cache not to invalidate itself for concurrently inserted data. ([Bug#28249](#))
- The full-text parser could enter an infinite loop if it encountered an illegal multi-byte sequence or a sequence that has no mapping to Unicode. ([Bug#29464](#))
- Failure to consider collation when comparing space characters could lead to incorrect index entry order, making it impossible to find some index values. ([Bug#29461](#))
- Several `InnoDB` assertion failures were corrected. ([Bug#25645](#))
- `InnoDB` displayed an incorrect error message when a `CREATE TABLE` statement exceeded the `InnoDB` maximum allowable row size. ([Bug#21101](#))

- InnoDB produced an unnecessary (and harmless) warning: `InnoDB: Error: trying to declare trx to enter InnoDB, but InnoDB: it already is declared.` (Bug#20090)
- Backup software can cause `ERROR_SHARING_VIOLATION` or `ERROR_LOCK_VIOLATION` conditions during file operations. InnoDB now retries forever until the condition goes away. (Bug#9709)
- In strict SQL mode, errors silently stopped the SQL thread even for errors named using the `--slave-skip-errors` option. (Bug#28839)
- MyISAM corruption could occur with the `cp932_japanese_ci` collation for the `cp932` character set due to incorrect comparison for trailing space. (Bug#29333)
- Searching a `FULLTEXT` index for a word with the boolean mode truncation operator could cause an infinite loop. (Bug#29445)
- `CHECK TABLE` could erroneously report table corruption for a `CSV` table if multiple threads were modifying the table at the same time. (Bug#29253)
- Clients using SSL could hang the server. (Bug#29579)
- Single-row inserts could report a row count greater than one. (Bug#29692)
- For a table with a `DATE` column `date_col` such that selecting rows with `WHERE date_col = 'date_val 00:00:00'` yielded a non-empty result, adding `GROUP BY date_col` caused the result to be empty. (Bug#29729)
- If a stored procedure was created and invoked prior to selecting a default database with `USE`, a `NO DATABASE SELECTED` error occurred. (Bug#28551)
- Indexing column prefixes in InnoDB tables could cause table corruption. (Bug#28138)
- `SHOW INNODB STATUS` caused an assertion failure under high load. (Bug#22819)
- If a slave timed out while registering with the master to which it was connecting, auto-reconnect failed thereafter. (Bug#19328)
- For the embedded server, the `mysql_stmt_store_result()` C API function caused a memory leak for empty result sets. (Bug#29687)
- `mysql-stress-test.pl` and `mysqld_multi.server.sh` were missing from some binary distributions. (Bug#21023, Bug#25486)
- `SELECT ... INTO OUTFILE` followed by `LOAD DATA` could result in garbled characters when the `FIELDS ENCLOSED BY` clause named a delimiter of `'0'`, `'b'`, `'n'`, `'r'`, `'t'`, `'N'`, or `'Z'` due to an interaction of character encoding and doubling for data values containing the enclosed-by character. (Bug#29294)
- A duplicate-key error message could display an incorrect key value when not all columns of the key were used to select rows for update. (Bug#28158)
- For some event-creation problems, the server displayed messages that implied the problems were errors when they were only warnings. (Bug#27406)
- Error returns from the `time()` system call were ignored. (Bug#27198)
- `ALTER DATABASE` did not require at least one option. (Bug#25859)
- Creation of a legal stored procedure could fail if no default database had been selected. (Bug#29050)
- `mysqld_safe` produced error messages and did not create the error log file under some circumstances. (Bug#29634)
- The thread ID was not reset properly after execution of `mysql_change_user()`, which could cause replication failure when replicating temporary tables. (Bug#29734)
- The `FEDERATED` storage engine failed silently for `INSERT ... ON DUPLICATE KEY UPDATE` if a duplicate key violation occurred. `FEDERATED` does not support `ON DUPLICATE KEY UPDATE`, so now it correctly returns an `ER_DUP_KEY` error if a duplicate key violation occurs. (Bug#25511)
- For a multiple-row insert into a `FEDERATED` table that refers to a remote transactional table, if the insert failed for a row due to constraint failure, the remote table would contain a partial commit (the rows preceding the failed one) instead of rolling back the

statement completely. This occurred because the rows were treated as individual inserts.

Now **FEDERATED** performs bulk-insert handling such that multiple rows are sent to the remote table in a batch. This provides a performance improvement and enables the remote table to perform statement rollback properly should an error occur. This capability has the following limitations:

- The size of the insert cannot exceed the maximum packet size between servers. If the insert exceeds this size, it is broken into multiple packets and the rollback problem can occur.
- Bulk-insert handling does not occur for **INSERT ... ON DUPLICATE KEY UPDATE**.

([Bug#25513](#))

- **ALTER VIEW** is not supported as a prepared statement but was not being rejected. **ALTER VIEW** is now prohibited as a prepared statement or when called within stored routines. ([Bug#28846](#))
- Calling **mysql_options()** after **mysql_real_connect()** could cause clients to crash. ([Bug#29247](#))
- If an **ENUM** column contained '' as one of its members (represented with numeric value greater than 0), and the column contained error values (represented as 0 and displayed as ''), using **ALTER TABLE** to modify the column definition caused the 0 values to be given the numeric value of the non-zero '' member. ([Bug#29251](#))
- Aggregations in subqueries that refer to outer query columns were not always correctly referenced to the proper outer query. ([Bug#27333](#))
- Use of **SHOW BINLOG EVENTS** for a non-existent log file followed by **PURGE MASTER LOGS** caused a server crash. ([Bug#29420](#))
- A number of unsupported constructs — including prohibited constructs, the **UCASE()** function, and nested function calls — were permitted in partitioning expressions. ([Bug#18198](#), [Bug#26082](#), [Bug#29308](#))
- **SHOW BINLOG EVENTS** displayed incorrect values of **End_log_pos** for events associated with transactional storage engines. ([Bug#22540](#))
- **mysqldump** created a stray file when a given a too-long filename argument. ([Bug#29361](#))
- The semantics of **BIGINT** depended on platform-specific characteristics. ([Bug#29079](#))
- **mysqld** failed to exit during shutdown. ([Bug#29133](#))
- Inserting a negative number into a **CSV** table could corrupt it. ([Bug#29353](#))
- Deleting from a **CSV** table could corrupt it. ([Bug#29411](#))
- For a statement of the form **CREATE t1 SELECT integer_constant**, the server created the column using the **DECIMAL** data type for large negative values that are within the range of **BIGINT**. ([Bug#28625](#))
- Runtime changes to the **log_queries_not_using_indexes** system variable were ignored. ([Bug#28808](#))
- Under ActiveState Perl, **mysql-test-run.pl** would not run. ([Bug#18415](#))
- Under ActiveState Perl, **mysql-test-run.pl** could kill itself when attempting to kill other processes. ([Bug#25657](#))
- Assertion failure could occur for grouping queries that employed **DECIMAL** user variables with assignments to them. ([Bug#29417](#))
- For **CAST(expr AS DECIMAL(M,D))**, the limits of 65 and 30 on the precision (*M*) and scale (*D*) were not enforced. ([Bug#29415](#))
- Corrupt data resulted from use of **SELECT ... INTO OUTFILE 'file_name' FIELDS ENCLOSED BY 'c'**, where *c* is a digit or minus sign, followed by **LOAD DATA INFILE 'file_name' FIELDS ENCLOSED BY 'c'**. ([Bug#29442](#))
- **AsText()** could fail with a buffer overrun. ([Bug#29166](#))
- Inserting into **InnoDB** tables and executing **RESET MASTER** in multiple threads cause assertion failure in debug server binaries. ([Bug#28983](#))

- The index merge union access algorithm could produce incorrect results with **InnoDB** tables. The problem could also occur for queries that used **DISTINCT**. (Bug#25798)
- Results for a select query that aliases the column names against a view could duplicate one column while omitting another. This bug could occur for a query over a multiple-table view that includes an **ORDER BY** clause in its definition. (Bug#29392)
- **gcov** coverage-testing information was not written if the server crashed. (Bug#29543)
- **FULLTEXT** indexes could be corrupted by certain **gbk** characters. (Bug#29299)
- **REPLACE**, **INSERT IGNORE**, and **UPDATE IGNORE** did not work for **FEDERATED** tables. (Bug#29019)
- **CHECK TABLE** for **ARCHIVE** tables could falsely report table corruption or cause a server crash. (Bug#29207)
- **SELECT ... FOR UPDATE** with partitioned tables could cause a server crash. (Bug#28026)
- Updates to rows in a partitioned table could update the wrong column. (Bug#26827)
- Updates to a **CSV** table could cause a server crash or update the table with incorrect values. (Bug#28971)
- Dropping a user-defined function could cause a server crash if the function was still in use by another thread. (Bug#27564)
- The server crashed when the size of an **ARCHIVE** table grew larger than 2GB. (Bug#15787)
- An assertion failure occurred if a query contained a conjunctive predicate of the form **view_column = constant** in the **WHERE** clause and the **GROUP BY** clause contained a reference to a different view column. The fix also enables application of an optimization that was being skipped if a query contained a conjunctive predicate of the form **view_column = constant** in the **WHERE** clause and the **GROUP BY** clause contained a reference to the same view column. (Bug#29104)
- Replication of partitioned tables using the **InnoDB** storage engine failed with **binlog-format=ROW** or **binlog-format=MIXED**. (Bug#28430)
- If a storage engine has its own logging capability, then any statement using both this engine and some other engine not having its own logging could not be correctly logged, due to the fact that entries from one engine could be logged before entries from the other engine were. This did not generate any error messages when it occurred.

Now, if multiple storage engines are used in a statement and at least one of them has its own logging capability, then an error message is generated and the statement is not executed. (Bug#28722)

Note

Currently, the only storage engine to have its own logging capability is **NDBCLUSTER**.

- Using the **READ COMMITTED** transaction isolation level caused mixed and statement-based replication to fail. (Bug#23051)
- The server returned data from **SHOW CREATE TABLE** statement or a **SELECT** statement on an **INFORMATION_SCHEMA** table using the **binary** character set. (Bug#10491)
- Mixing binary and **utf8** columns in a union caused field lengths to be calculated incorrectly, resulting in truncation. (Bug#29205)
- **LOCK TABLES** was not atomic when more than one **InnoDB** tables were locked. (Bug#29154)
- Queries that performed a lookup into a **BINARY** index containing key values ending with spaces caused an assertion failure for debug builds and incorrect results for non-debug builds. (Bug#29087)
- Selecting a column not present in the selected-from table caused an extra error to be produced by **SHOW ERRORS**. (Bug#28677)
- If an **INSERT INTO ... SELECT** statement inserted into the same table that the **SELECT** retrieved from, and the **SELECT** included **ORDER BY** and **LIMIT** clauses, different data was inserted than the data produced by the **SELECT** executed by itself. (Bug#29095)
- On 64-bit Windows systems, the Config Wizard failed to complete the setup because 64-bit Windows does not resolve dynamic linking of the 64-bit **libmysql.dll** to a 32-bit application like the Config Wizard. (Bug#14649)
- For a join with **GROUP BY** and/or **ORDER BY** and a view reference in the **FROM** list, the query metadata erroneously showed empty table aliases and database names for the view columns. (Bug#28898)

- For a `ucs2` column, `GROUP_CONCAT()` did not convert separators to the result character set before inserting them, producing a result containing a mixture of two different character sets. (Bug#28925)
- Index-based range reads could fail for comparisons that involved contraction characters (such as `ch` in Czech or `ll` in Spanish). (Bug#27345)
- The Windows implementation of `pthread_join()` was incorrect and could cause crashes. (Bug#26564)
- Sort order of the collation wasn't used when comparing trailing spaces. This could lead to incorrect comparison results, incorrectly created indexes, or incorrect result set order for queries that include an `ORDER BY` clause. (Bug#29261)
- Many threads accessing a `CSV` table simultaneously could cause an assertion failure. (Bug#29252)
- `mysqlbinlog --hexdump` generated incorrect output due to omission of the '#' comment character for some comment lines. (Bug#28293)
- Index creation could corrupt the table definition in the `.frm` file: 1) A table with the maximum number of key segments and maximum length key name would have a corrupted `.frm` file, due to incorrect calculation of the total key length. 2) `MyISAM` would reject a table with the maximum number of keys and the maximum number of key segments in all keys. (It would allow one less than this total maximum.) Now `MyISAM` accepts a table defined with the maximum. (Bug#26642)
- Dropping the definer of an active event caused the server to crash. (Bug#28924)
- Executing `ALTER EVENT` on an event whose definer's event creation privileges had been revoked cause the server to crash. (Bug#28873)
- Creating an event using `ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL ...` could in some cases cause `mysqld` to crash. (Bug#28881)
- Under some circumstances, a `SELECT ... FROM mysql.event` could cause the server to crash. (Bug#29156)
- Some functions when used in partitioning expressions could cause `mysqld` to crash. (Bug#27084)
- The `SUBSTRING()` function returned the the entire string instead of an empty string when it was called from a stored procedure and when the length parameter was specified by a variable with the value '0'. (Bug#27130)
- The `LOCATE()` function returned `NULL` if any of its arguments evaluated to `NULL`. Likewise, the predicate, `LOCATE(str, NULL) IS NULL`, erroneously evaluated to `FALSE`. (Bug#27932)
- A query with `DISTINCT` in the select list to which the loose-scan optimization for grouping queries was applied returned an incorrect result set when the query was used with the `SQL_BIG_RESULT` option. (Bug#25602)
- A network structure was initialized incorrectly, leading to embedded server crashes. (Bug#29117)
- Fixed a case of unsafe aliasing in the source that caused a client library crash when compiled with `gcc 4` at high optimization levels. (Bug#27383)

C.1.5. Changes in release 5.1.20 (25 June 2007)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- **Incompatible change:** `mysqld_safe` now supports error logging to `syslog` on systems that support the `logger` command.

The new `--syslog` and `--skip-syslog` options can be used instead of the `--log-error` option to control logging behavior, as described in [Section 5.3.1, “mysqld_safe — MySQL Server Startup Script”](#). The default is to use `syslog`, which differs from the previous default behavior of writing an error log file. ([Bug#4858](#))

Currently, logging to `syslog` may fail to operate correctly in some cases, so we recommend that you use `--skip-syslog` or `--log-error`. To maintain the older behavior if you were using no error-logging option, use `--skip-syslog`. If you were using `--log-error`, continue to use it.

Note: In 5.1.21, the default is changed to `--skip-syslog`, which is compatible with releases prior to 5.1.20.

- **Incompatible change:** It is no longer possible to partition the log tables. ([Bug#27816](#))
- User variables and stored procedure variables are now supported for use in XPath expressions employed as arguments to the `ExtractValue()` and `UpdateXML()` functions. ([Bug#26518](#))

This means that:

- XPath can now be used to load data from XML files using virtually any format, and so able to import data from most third party software which either has XML export functionality, or uses XML natively as a storage format.
- Various complex conditions can be put on rows and columns, so one can filter for desired rows (or skip unwanted rows) when loading XML.
- Various types of preprocessing using SQL functions are now possible when loading XML. For example, you can concatenate two XML tag or attribute values into a single column value using `CONCAT()`, or remove some parts of the data using `REPLACE()`.

See [Section 11.10, “XML Functions”](#), for more information.

- **NDB Cluster:** A new configuration parameter `ODirect` causes **NDB** to attempt using `O_DIRECT` writes for LCP, backups, and redo logs, often lowering CPU usage.
- **NDB Cluster:** The server source tree now includes scripts to simplify building MySQL with SCI support. For more information about SCI interconnects and these build scripts, see [Section 16.12.1, “Configuring MySQL Cluster to use SCI Sockets”](#). ([Bug#25470](#))
- **NDB Cluster:** `auto_increment_increment` and `auto_increment_offset` are now supported for **NDB** tables. ([Bug#26342](#))
- **NDB Cluster:** The cluster management client now stores command history between sessions. ([Bug#29073](#))
- **NDB Cluster:** `ndb_error_reporter` now preserves timestamps on files. ([Bug#29074](#))
- **NDB Cluster:** The `TimeBetweenWatchdogCheckInitial` configuration parameter was added to allow setting of a separate watchdog timeout for memory allocation during startup of the data nodes. See [Section 16.4.4.5, “Defining Data Nodes”](#), for more information. ([Bug#28899](#))
- **NDB Cluster:** It is now possible to set the maximum size of the allocation unit for table memory using the `MaxAllocate` configuration parameter. ([Bug#29044](#))
- If a **MERGE** table cannot be opened or used because of a problem with an underlying table, `CHECK TABLE` now displays information about which table caused the problem. ([Bug#26976](#))
- The `SQL_MODE`, `FOREIGN_KEY_CHECKS`, `UNIQUE_CHECKS`, character set/collations, and `SQL_AUTO_IS_NULL` session variables are written to the binary log and honoured during replication. See [Section 5.11.4, “The Binary Log”](#).
- Binary distributions for some platforms did not include shared libraries; now shared libraries are shipped for all platforms except AIX 5.2 64-bit. Exception: The library for the `libmysqld` embedded server is not shared except on Windows. ([Bug#13450](#), [Bug#16520](#), [Bug#26767](#))

Bugs fixed:

- **Security fix:** A malformed password packet in the connection protocol could cause the server to crash. Thanks for Dormando for reporting this bug and providing details and a proof of concept. ([CVE-2007-3780](#), [Bug#28984](#))

- **Security Fix:** `CREATE TABLE LIKE` did not require any privileges on the source table. Now it requires the `SELECT` privilege. (CVE-2007-3781, Bug#25578)

In addition, `CREATE TABLE LIKE` was not isolated from alteration by other connections, which resulted in various errors and incorrect binary log order when trying to execute concurrently a `CREATE TABLE LIKE` statement and either DDL statements on the source table or DML or DDL statements on the target table. (Bug#23667)

- **Incompatible change:** When `mysqldump` was run with the `--delete-master-logs` option, binary log files were deleted before it was known that the dump had succeeded, not after. (The method for removing log files used `RESET MASTER` prior to the dump. This also reset the binary log sequence numbering to `.000001`.) Now `mysqldump` flushes the logs (which creates a new binary log number with the next sequence number), performs the dump, and then uses `PURGE MASTER LOGS` to remove the log files older than the new one. This also preserves log numbering because the new log with the next number is generated and only the preceding logs are removed. However, this may affect applications if they rely on the log numbering sequence being reset. (Bug#24733)
- **Incompatible change:** Some error codes had error numbers in MySQL 5.1 different from the numbers in MySQL 5.0. In MySQL 5.1, error numbers have been changed to match the MySQL 5.0 values: Error codes with value of 1458 or higher have changed in MySQL 5.1 now. Client applications designed to work with MySQL 5.1 with hard-coded error code values (for example, in statements such as `if (mysql_errno(mysql) == 1463) { ... }`) need to be updated in the source code. All clients designed to work with MySQL 5.1 that test error codes (for example, in statements such as `if (mysql_errno(mysql) == ER_VIEW_RECURSIVE) { ... }`) should be recompiled. Existing 5.0 clients should now work, without changes or recompilation, against servers for MySQL 5.1.20 or higher. (Bug#29245)
- **Incompatible change:** The use of an `ORDER BY` or `DISTINCT` clause with a query containing a call to the `GROUP_CONCAT()` function caused results from previous queries to be redisplayed in the current result. The fix for this includes replacing a `BLOB` value used internally for sorting with a `VARCHAR`. This means that for long results (more than 65,535 bytes), it is possible for truncation to occur; if so, an appropriate warning is issued. (Bug#23856, Bug#28273)
- **NDB Cluster:** A query having a large `IN(...)` or `NOT IN(...)` list in the `WHERE` condition on an `NDB` table could cause `mysqld` to crash. (Bug#29185)
- **NDB Cluster** (Disk Data): When dropping a page, the stack's bottom entry could sometime be left “cold” rather than “hot”, violating the rules for stack pruning. (Bug#29176)
- **NDB Cluster:** In the event that two data nodes in the same node group and participating in a GCP crashed before they had written their respective `P0.sysfile` files, `QMGR` could refuse to start, issuing an invalid `INSUFFICIENT NODES FOR RESTART` error instead. (Bug#29167)
- **NDB Cluster:** Memory corruption could occur due to a problem in the `DBTUP` kernel block. (Bug#29229)
- **NDB Cluster** (APIs): An invalid error code could be set on transaction objects by `BLOB` handling code. (Bug#28724)
- **NDB Cluster:** Attempting to restore a `NULL` row to a `VARBINARY` column caused `ndb_restore` to fail. (Bug#29103)
- **NDB Cluster:** `ndb_mgm` could hang when connecting to a nonexistent host. (Bug#28847)
- **NDB Cluster** (APIs): The timeout set using the MGM API `ndb_mgm_set_timeout()` function was incorrectly interpreted as seconds rather than as milliseconds. (Bug#29063)
- **NDB Cluster:** When shutting down `mysqld`, the `NDB` binlog process was not shut down before log cleanup began. (Bug#28949)
- **NDB Cluster:** A race condition could result when non-master nodes (in addition to the master node) tried to update active status due to a local checkpoint. Now only the master updates the active status. (Bug#28717)
- **NDB Cluster:** The management client's response to `START BACKUP WAIT COMPLETED` did not include the backup ID. (Bug#27640)
- **NDB Cluster:** A regression in the heartbeat monitoring code could lead to node failure under high load. This issue affected MySQL 5.1.19 only. (Bug#28783)
- **NDB Cluster** (Replication): When replicating `MyISAM` or `InnoDB` tables to a MySQL Cluster, it was not possible to determine exactly what had been applied following a shutdown of the slave cluster or `mysqld` process. (Bug#26783)
- **NDB Cluster:** Having large amounts of memory locked caused swapping to disk. (Bug#28751)
- **NDB Cluster:** The actual value of `MaxNoOfOpenFiles` as used by the cluster was offset by 1 from the value set in `con -`

`fig.ini`. This meant that setting `InitialNoOpenFiles` to the same value always caused an error. (Bug#28749)

- **NDB Cluster**: LCP files were not removed following an initial system restart. (Bug#28726)
- **NDB Cluster**: A fast global checkpoint under high load with a high usage of the redo buffer caused data nodes to fail. (Bug#28653)
- **NDB Cluster** (Disk Data): When loading data into a cluster following a version upgrade, the data nodes could forcibly shut down due to page and buffer management failures. (Bug#28525)
- **NDB Cluster**: `UPDATE IGNORE` statements involving the primary keys of multiple tables could result in data corruption. (Bug#28719)
- **NDB Cluster**: A corrupt schema file could cause a `FILE ALREADY OPEN` error. (Bug#28770)
- **NDB Cluster** Disk Data: Repeated `INSERT` and `DELETE` operations on a Disk Data table having one or more large `VARCHAR` columns could cause data nodes to fail. (Bug#20612)
- **NDB Cluster** (Replication): A replicated unique key allowed duplicate key inserts on the slave. (Bug#27044)
- In `SHOW SLAVE STATUS` output, `Last_Errno` and `Last_Error` were not set after `master_retry_count` errors had occurred. To provide additional information, the statement now displays four additional columns:
 - `Last_IO_Errno`: The number of the last error that caused the I/O thread to stop
 - `Last_IO_Error`: A description of the last error that caused the I/O thread to stop
 - `Last_SQL_Errno`: The number of the last error that caused the SQL thread to stop
 - `Last_SQL_Error`: A description of the last error that caused the SQL thread to stop

Also, `Last_Errno` and `Last_Error` now are aliases for `Last_SQL_Errno` and `Last_SQL_Error`. (Bug#24954)

- `DROP USER` statements that named multiple users, only some of which could be dropped, were replicated incorrectly. (Bug#29030)
- It was possible to set `SQL_SLAVE_SKIP_COUNTER` such that the slave would jump into the middle of an event group. (Bug#28618)
- On some systems, `udf_example.c` returned an incorrect result length. Also on some systems, `mysql-test-run.pl` could not find the shared object built from `udf_example.c`. (Bug#27741)
- The server made strong assumptions about the structure of the `general_log` and `slow_log` log tables: It supported only the table structure defined in the `mysql` database creation scripts. The server also allowed limited `ALTER TABLE` operations on the log tables, but adding an `AUTO_INCREMENT` column did not properly initialize the column, and subsequent inserts into the table could fail to generate correct sequence numbers. Now an `ALTER TABLE` statement that adds an `AUTO_INCREMENT` column populates the column correctly. In addition, when the server writes a log table row, it will set columns not present in the original table structure to their default values. (Bug#27857)
- The embedded server library displayed error messages at startup if the `mysql.plugin` table was not present. This no longer occurs. (Bug#25800)
- The `-lmtalloc` library was removed from the output of `mysql_config` on Solaris, as it caused problems when building `DBD:mysql` (and possibly other applications) on that platform that tried to use `dlopen()` to access the client library. (Bug#18322)
- Embedded `/* ... */` comments were handled incorrectly within the definitions of stored programs and views, resulting in malformed definitions. This also affected binary log contents. (Bug#25411, Bug#26302)
- The `binlog_format` system variable value was empty if the server was started with binary logging disabled. Now it is set to `MIXED`. (Bug#28752)
- Conversion of U+00A5 YEN SIGN and U+203E OVERLINE from `ucs2` to `ujis` produced incorrect results. (Bug#28600)
- Some valid identifiers were not parsed correctly. (Bug#28127)
- On Windows, `USE_TLS` was not defined for `mysqlclient.lib`. (Bug#28860)

- A too-long `shared-memory-base-name` value could cause a buffer overflow and crash the server or clients. (Bug#24924)
- On Windows, an application that called `mysql_thread_init()` but forgot to call `mysql_thread_end()` would get this error: `Error in my_thread_global_end()` (Bug#25621)
- A stack overrun could occur when storing `DATETIME` values using repeated prepared statements. (Bug#27592)
- `INSERT ... ON DUPLICATE KEY UPDATE` could under some circumstances silently update rows when it should not have. (Bug#28904)
- Using events in replication could cause the slave to crash. (Bug#28953)
- `ALTER TABLE ... ENABLE KEYS` could cause `mysqld` to crash when executed on a table containing on a `MyISAM` table containing billions of rows. (Bug#27029)
- A `FLUSH TABLES WITH READ LOCK` statement followed by a `FLUSH LOGS` statement caused a deadlock if the general log or the slow query log was enabled. (Bug#26380)
- The `TRUNCATE` statement was handled differently by the server when row-based logging was in effect, even though the binlogging format in effect does not effect the fact that `TRUNCATE` is always logged as a statement. (Bug#29130)
- When the query cache was fully used, issuing `RENAME DATABASE` or `RENAME SCHEMA` could cause the server to hang, with 100% CPU usage. (Bug#28211)
- Binary content `0x00` in a `BLOB` column sometimes became `0x5C 0x00` following a dump and reload, which could cause problems with data using multi-byte character sets such as `GBK` (Chinese). This was due to a problem with `SELECT INTO OUTFILE` whereby `LOAD DATA` later incorrectly interpreted `0x5C` as the second byte of a multi-byte sequence rather than as the `SOLIDUS` (“\”) character, used by MySQL as the escape character. (Bug#26711)
- If one of the queries in a `UNION` used the `SQL_CACHE` option and another query in the `UNION` contained a nondeterministic function, the result was still cached. For example, this query was incorrectly cached:

```
SELECT NOW() FROM t1 UNION SELECT SQL_CACHE 1 FROM t1;
```

(Bug#29053)

- Queries using UDFs or stored functions were cached. (Bug#28921)
- The modification of a table by a partially completed multi-column update was not recorded in the binlog, rather than being marked by an event and a corresponding error code. (Bug#27716)
- `SHOW ENGINES` and queries on `INFORMATION_SCHEMA.ENGINES` did not use the same values for representing the same storage engine states. (Bug#27684)
- Connections from one `mysqld` server to another failed on Mac OS X, affecting replication and `FEDERATED` tables. (Bug#26664)
- The “manager thread” of the LinuxThreads implementation was unintentionally started before `mysqld` had dropped privileges (to run as an unprivileged user). This caused signaling between threads in `mysqld` to fail when the privileges were finally dropped. (Bug#28690)
- A query that grouped by the result of an expression returned a different result when the expression was assigned to a user variable. (Bug#28494)
- The result of evaluation for a view's `CHECK OPTION` option over an updated record and records of merged tables was arbitrary and dependant on the order of records in the merged tables during the execution of the `SELECT` statement. (Bug#28716)
- Outer join queries with `ON` conditions over constant outer tables did not return `NULL`-complemented rows when conditions were evaluated to `FALSE`. (Bug#28571)
- An update on a multiple-table view with the `CHECK OPTION` clause and a subquery in the `WHERE` condition could cause an assertion failure. (Bug#28561)
- `mysql_affected_rows()` could return an incorrect result for `INSERT ... ON DUPLICATE KEY UPDATE` if the `CLIENT_FOUND_ROWS` flag was set. (Bug#28505)
- Using `ALTER TABLE` to move columns resulted only in the columns being renamed. The table contents were not changed.

[Bug#28427](#))

- Storing a large number into a [FLOAT](#) or [DOUBLE](#) column with a fixed length could result in incorrect truncation of the number if the column's length was greater than 31. ([Bug#28121](#))
- [HASH](#) indexes on [VARCHAR](#) columns with binary collations did not ignore trailing spaces from strings before comparisons. This could result in duplicate records being successfully inserted into a [MEMORY](#) table with unique key constraints. A consequence was that internal [MEMORY](#) tables used for [GROUP BY](#) calculation contained duplicate rows that resulted in duplicate-key errors when converting those temporary tables to [MyISAM](#), and that error was incorrectly reported as a [table is full](#) error. ([Bug#27643](#))
- The server deducted some bytes from the [key_cache_block_size](#) option value and reduced it to the next lower 512 byte boundary. The resulting block size was not a power of two. Setting the [key_cache_block_size](#) system variable to a value that is not a power of two resulted in [MyISAM](#) table corruption. ([Bug#23068](#), [Bug#25853](#), [Bug#28478](#))
- [ON](#) conditions from [JOIN](#) expressions were ignored when checking the [CHECK OPTION](#) clause while updating a multiple-table view that included such a clause. ([Bug#27827](#))
- The [IS_UPDATABLE](#) column in the [INFORMATION_SCHEMA.VIEWS](#) table was not always set correctly. ([Bug#28266](#))
- For [CAST\(\)](#) of a [NULL](#) value with type [DECIMAL](#), the return value was incorrectly initialized, producing a runtime error for binaries built using Visual C++ 2005. ([Bug#28250](#))
- [DECIMAL](#) values beginning with nine 9 digits could be incorrectly rounded. ([Bug#27984](#))
- For debug builds, [ALTER TABLE](#) could trigger an assertion failure due to occurrence of a deadlock when committing changes. ([Bug#28652](#))
- Searches on indexed and non-indexed [ENUM](#) columns could return different results for empty strings. ([Bug#28729](#))
- Non-[utf8](#) characters could get mangled when stored in [CSV](#) tables. ([Bug#28862](#))
- If a stored function or trigger was killed, it aborted but no error was thrown, allowing the calling statement to continue without noticing the problem. This could lead to incorrect results. ([Bug#27563](#))
- When [ALTER TABLE](#) was used to add a new [DATE](#) column with no explicit default value, ['0000-00-00'](#) was used as the default even if the SQL mode included the [NO_ZERO_DATE](#) mode to prohibit that value. A similar problem occurred for [DATETIME](#) columns. ([Bug#27507](#))
- Statements within triggers ignored the value of the [low_priority_updates](#) system variable. ([Bug#26162](#))
- The server was ignoring the return value of the [parse\(\)](#) function for full-text parser plugins. ([Bug#18839](#))
- Queries that used [UUID\(\)](#) were incorrectly allowed into the query cache. (This should not happen because [UUID\(\)](#) is non-deterministic.) ([Bug#28897](#))
- The [Bytes_received](#) and [Bytes_sent](#) status variables could hold only 32-bit values (not 64-bit values) on some platforms. ([Bug#28149](#))
- [SHOW GLOBAL VARIABLES](#) repeated some variable names. ([Bug#28580](#))
- Passing a [DECIMAL](#) value as a parameter of a statement prepared with [PREPARE](#) resulted in an error. ([Bug#28509](#))
- For attempts to open a non-existent table, the server should report [ER_NO_SUCH_TABLE](#) but sometimes reported [ER_TABLE_NOT_LOCKED](#). ([Bug#27907](#))
- Due to a race condition, executing [FLUSH PRIVILEGES](#) in one thread could cause brief table unavailability in other threads. ([Bug#24988](#))
- Conversion errors could occur when constructing the condition for an [IN](#) predicate. The predicate was treated as if the affected column contains [NULL](#), but if the [IN](#) predicate is inside [NOT](#), incorrect results could be returned. ([Bug#22855](#))
- Linux binaries were unable to dump core after executing a [setuid\(\)](#) call. ([Bug#21723](#))
- Using up-arrow for command-line recall in [mysql](#) could cause a segmentation fault. ([Bug#10218](#))
- Long pathnames for internal temporary tables could cause stack overflows. ([Bug#29015](#))

- If a program binds a given number of parameters to a prepared statement handle and then somehow changes `stmt->param_count` to a different number, `mysql_stmt_execute()` could crash the client or server. (Bug#28934)
- Using a `VIEW` created with a non-existing `DEFINER` could lead to incorrect results under some circumstances. (Bug#28895)
- In MySQL 5.1.15, a new error code `ER_DUP_ENTRY_WITH_KEY_NAME` (1582) was introduced to replace `ER_DUP_ENTRY` (1062) so that the key name could be provided instead of the key number. This was unnecessary, so `ER_DUP_ENTRY` is used again and the key name is printed. The incompatibility introduced in 5.1.15 no longer applies. (Bug#28842)
- When one thread attempts to lock two (or more) tables and another thread executes a statement that aborts these locks (such as `REPAIR TABLE`, `OPTIMIZE TABLE`, or `CHECK TABLE`), the thread might get a table object with an incorrect lock type in the table cache. The result is table corruption or a server crash. (Bug#28574)
- The server crashed when attempting to open a table having a `#mysql50#` prefix in the database or table name. The server now will not open such tables. (This prefix is reserved by `mysql_upgrade` for accessing 5.0 tables that have names not yet encoded for 5.1.) (Bug#26402)
- Running `SHOW TABLE STATUS` while performing a high number of inserts on partitioned tables with a great many partitions could cause the server to crash. (Bug#28806)
- An error occurred trying to connect to `mysqld-debug.exe`. (Bug#27597)
- Using an `INTEGER` column from a table to `ROUND()` a number produced different results than using a constant with the same value as the `INTEGER` column. (Bug#28980)
- The `PARTITION_COMMENT` column of the `INFORMATION_SCHEMA.PARTITIONS` table had the wrong default value. (Bug#28007)
- Performing `ALTER TABLE ... ADD PARTITION` or `ALTER TABLE DROP PARTITION` could result in inconsistent data, or cause the server to crash, if done concurrently with other accesses to the table. (Bug#28477, Bug#28488)
- InnoDB tables using an indexed `CHAR` column with `utf8` as the default character set could fail to return the right rows. (Bug#28878)
- Using `BETWEEN` with non-indexed date columns and short formats of the date string could return incorrect results. (Bug#28778)
- When using a `MEMORY` table on Mac OS X, dropping a table and then creating a table with the same name could cause the information of the deleted table to remain accessible, leading to index errors. (Bug#28309)
- Granting access privileges to an individual table where the database or table name contained an underscore would fail. (Bug#18660)
- A subquery with `ORDER BY` and `LIMIT 1` could cause a server crash. (Bug#28811)
- Selecting `GEOMETRY` columns in a `UNION` caused a server crash. (Bug#28763)
- `mysqltest` used a too-large stack size on PPC/Debian Linux, causing thread-creation failure for tests that use many threads. (Bug#28333)
- When constructing the path to the original `.frm` file, `ALTER ... RENAME` was unnecessarily (and incorrectly) lowercasing the entire path when not on a case-insensitive filesystem, causing the statement to fail. (Bug#28754)
- `PURGE MASTER LOGS BEFORE (subquery)` caused a server crash. Subqueries are forbidden in the `BEFORE` clause now. (Bug#28553)
- A server crash could happen under rare conditions such that a temporary table outgrew heap memory reserved for it and the remaining disk space was not big enough to store the table as a `MyISAM` table. (Bug#28449)
- On some Linux distributions where LinuxThreads and NPTL `glibc` versions both are available, statically built binaries can crash because the linker defaults to LinuxThreads when linking statically, but calls to external libraries (such as `libnss`) are resolved to NPTL versions. This cannot be worked around in the code, so instead if a crash occurs on such a binary/OS combination, print an error message that provides advice about how to fix the problem. (Bug#24611)
- Stack overflow caused server crashes. (Bug#21476)
- The test case for `ysqldump` failed with `bin-log` disabled. (Bug#28372)

- The `check-cpu` script failed to detect AMD64 Turion processors correctly. (Bug#17707)
- Attempting to create an index on a `BIT` column failed after modifying the column. (Bug#28631)
- After an upgrade, the names of stored routines referenced by views were no longer displayed by `SHOW CREATE VIEW`. This was a regression introduced by the fix for Bug#23491. (Bug#28605)
- Killing from one connection a long-running `EXPLAIN QUERY` started from another connection caused `mysqld` to crash. (Bug#28598)
- When upgrading from MySQL 5.1.17 to 5.1.18, `mysql_upgrade` and `mysql_fix_privilege_tables` did not upgrade the system tables relating to the Event Scheduler correctly. (Bug#28521)
- Subselects returning `LONG` values in MySQL versions later than 5.0.24a returned `LONGLONG` prior to this. The previous behavior was restored. This issue was introduced by the fix for Bug#19714. (Bug#28492)
- Executing `EXPLAIN EXTENDED` on a query using a derived table over a grouping subselect could lead to a server crash. This occurred only when materialization of the derived tables required creation of an auxiliary temporary table, an example being when a grouping operation was carried out with usage of a temporary table. (Bug#28728)
- Binary logging of prepared statements could produce syntactically incorrect queries in the binary log, replacing some parameters with variable names rather than variable values. This could lead to incorrect results on replication slaves. (Bug#12826, Bug#26842)
- Attempting to `LOAD_FILE` from an empty floppy drive under Windows, caused the server to hang. For example, if you opened a connection to the server and then issued the command `SELECT LOAD_FILE('a:test');`, with no floppy in the drive, the server was inaccessible until the modal pop-up dialog box was dismissed. (Bug#28366)
- `mysqldump` calculated the required memory for a hex-blob string incorrectly causing a buffer overrun. This in turn caused `mysqldump` to crash silently and produce incomplete output. (Bug#28522)
- The query `SELECT '2007-01-01' + INTERVAL column_name DAY FROM table_name` caused `mysqld` to fail. (Bug#28450)
- The result of executing of a prepared statement created with `PREPARE s FROM "SELECT 1 LIMIT ?"` was not replicated correctly. (Bug#28464)
- For a given user variable `@v`, the statements `SELECT @v` and `CREATE TABLE ... AS SELECT @v` did not return the same data type. (Bug#26277)
- Trying to shut down the server following a failed `LOAD DATA INFILE` caused `mysqld` to crash. (Bug#17233)
- A number of `SHOW` statements caused `mysqld` to crash on recent versions of Solaris. This issue is believed to be present only in MySQL 5.1.12 and later. (Bug#23810)
- Using `CREATE TABLE LIKE ...` would raise an assertion when replicated to a slave. (Bug#18950)
- Forcing the use of an index on a `SELECT` query when the index had been disabled would raise an error without running the query. The query now executes, with a warning generated noting that the use of a disabled index has been ignored. (Bug#28476)
- When using `mysqlbinlog` with `--read-from-remote-server` to load the data direct from a remote MySQL server would cause a core dump when dumping certain binary log events. (Bug#17654)
- When using transactions and replication, shutting down the master in the middle of a transaction would cause all slaves to stop replicating. (Bug#22725)
- Recreating a view that already exists on the master would cause a replicating slave to terminate replication with a 'different error message on slave and master' error. (Bug#28244)
- A stored program that uses a variable name containing multibyte characters could fail to execute. (Bug#27876)
- The internal functions for table preparation, creation, and alteration were not re-execution friendly, causing problems in code that: repeatedly altered a table; repeatedly created and dropped a table; opened and closed a cursor on a table, altered the table, and then reopened the cursor; used `ALTER TABLE` to change a table's current `AUTO_INCREMENT` value; created indexes on `utf8` columns. (Bug#4968, Bug#6895, Bug#19182, Bug#19733, Bug#22060, Bug#24879)
- The query `SELECT /*2*/ user, host, db, info FROM INFORMATION_SCHEMA.PROCESSLIST WHERE`

```
(command!='Daemon' || user='event_scheduler') AND (info IS NULL OR info NOT LIKE '%processlist%') ORDER BY INFO
```

yielded inconsistent results. (Bug#26338)

- Sending debugging information from a dump of the Event Scheduler to `COM_DEBUG` could cause the server to crash. (Bug#28075)
- Setting an interval of `EVERY 0 SECOND` for a scheduled event caused the server to crash. (Bug#28666)
- Following an invalid call to `UpdateXML()`, calling the function again (even if valid) crashed the server. (Bug#27898)
- Calling the `UpdateXML()` function using invalid XPath syntax caused memory corruption possibly leading to a crash of the server. (Bug#28558)

C.1.6. Changes in release 5.1.19 (25 May 2007)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- `INSERT DELAYED` statements for `BLACKHOLE` tables caused a server crash. The `BLACKHOLE` storage engine now supports `INSERT DELAYED`. (Bug#27998)
- The `BLACKHOLE` storage engine now supports `LOCK TABLES` and `UNLOCK TABLES`. (Bug#26241)
- The data type used for the `VARIABLE_VALUE` column of the following `INFORMATION_SCHEMA` tables has been changed to `VARCHAR`:
 - `GLOBAL_STATUS`
 - `SESSION_STATUS`
 - `GLOBAL_VARIABLES`
 - `SESSION_VARIABLES`For more information, see [Section 23.24, “The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables”](#), [Section 23.25, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”](#), and [Bug#26994](#)
- A new status variable, `Com_call_procedure`, indicates the number of calls to stored procedures. (Bug#27994)

Bugs fixed:

- **Security fix:** Use of a view could allow a user to gain update privileges for tables in other databases. (CVE-2007-3782, Bug#27878)
- **Security fix:** UDFs are supposed to be loadable only from the plugin directory, but this restriction was not being enforced. (Bug#28341)
- Comparing a `DATETIME` column value with a user variable yielded incorrect results. (Bug#28261)
- Comparison of the string value of a date showed as unequal to `CURTIME()`. Similar behavior was exhibited for `DATETIME` values. (Bug#28208)
- A buffer overflow could occur when using `DECIMAL` columns on Windows operating systems. (Bug#28361)

- **NDB Cluster**: When an API node sent more than 1024 signals in a single batch, **NDB** would process only the first 1024 of these, and then hang. ([Bug#28443](#))
- **NDB Cluster** (Disk Data): DDL operations were not supported on a partially started cluster. ([Bug#24631](#))
- **NDB Cluster**: A delay in obtaining **AUTO_INCREMENT** IDs could lead to excess temporary errors. ([Bug#28410](#))
- **NDB Cluster**: Local checkpoint files related to dropped **NDB** tables were not removed. ([Bug#28348](#))
- **NDB Cluster**: A failure to release internal resources following an error could lead to problems with single user mode. ([Bug#25818](#))
- **NDB Cluster**: Multiple operations involving deletes followed by reads were not handled correctly. ([Bug#28276](#))

Note

This issue could also affect MySQL Cluster Replication.

- **NDB Cluster** (Disk Data): Extremely large inserts into Disk Data tables could lead to data node failure in some circumstances. ([Bug#27942](#))
- **NDB Cluster**: Repeated insertion of data generated by **mysqldump** into **NDB** tables could eventually lead to failure of the cluster. ([Bug#27437](#))
- **NDB Cluster**: Restarting a data node caused SQL nodes to log repeatedly and unnecessarily the status of the event buffer. ([Bug#27292](#))

(This issue was known to occur in MySQL 5.1.16 and later only.)
- **NDB Cluster**: **ndb_mgmd** failed silently when the cluster configuration file contained invalid **[TCP]** entries. ([Bug#27207](#))
- **NDB Cluster**: **ndb_connectstring** did not appear in the output of **SHOW VARIABLES**. ([Bug#26675](#))
- **NDB Cluster** (APIs): In a multi-operation transaction, a delete operation followed by the insertion of an implicit **NULL** failed to overwrite an existing value. ([Bug#20535](#))
- Selecting **MIN()** on an indexed column that contained only **NULL** values caused **NULL** to be returned for other result columns. ([Bug#27573](#))
- **mysql_upgrade** failed if certain SQL modes were set. Now it sets the mode itself to avoid this problem. ([Bug#28401](#))
- Some test suite files were missing from some MySQL-test packages. ([Bug#26609](#))
- When dumping procedures, **mysqldump --compact** generated output that restored the session variable **SQL_MODE** without first capturing it. When dumping routines, **mysqldump --compact** neither set nor retrieved the value of **SQL_MODE**. ([Bug#28223](#))
- The second execution of a prepared statement from a **UNION** query with **ORDER BY RAND()** caused the server to crash. This problem could also occur when invoking a stored procedure containing such a query. ([Bug#27937](#))
- **CURDATE()** is less than **NOW()**, either when comparing **CURDATE()** directly (**CURDATE() < NOW()** is true) or when casting **CURDATE()** to **DATE(CAST(CURDATE() AS DATE) < NOW()** is true). However, storing **CURDATE()** in a **DATE** column and comparing **col_name < NOW()** incorrectly yielded false. This is fixed by comparing a **DATE** column as **DATETIME** for comparisons to a **DATETIME** constant. ([Bug#21103](#))
- For dates with 4-digit year parts less than 200, an incorrect implicit conversion to add a century was applied for date arithmetic performed with **DATE_ADD()**, **DATE_SUB()**, **+** **INTERVAL**, and **-** **INTERVAL**. (For example, **DATE_ADD('0050-01-01 00:00:00', INTERVAL 0 SECOND)** became **'2050-01-01 00:00:00'**.) ([Bug#18997](#))
- The result for **CAST()** when casting a value to **UNSIGNED** was limited to the maximum signed **BIGINT** value, not the maximum unsigned value. ([Bug#8663](#))
- Running **CHECK TABLE** concurrently with a **SELECT**, **INSERT** or other statement on Windows could corrupt a MyISAM table. ([Bug#25712](#))
- The error message for error number **137** did not report which database/table combination reported the problem. ([Bug#27173](#))

- Changing the size of a key buffer that is under heavy use could cause a server crash. The fix partially removes the limitation that `LOAD INDEX INTO CACHE` fails unless all indexes in a table have the same block size. Now the statement fails only if `IGNORE LEAVES` is specified. ([Bug#17332](#))
- `EXPLAIN` for a query on an empty table immediately after its creation could result in a server crash. ([Bug#28272](#))
- Grouping queries with correlated subqueries in `WHERE` conditions could produce incorrect results. ([Bug#28337](#))
- `libmysql.dll` could not be dynamically loaded on Windows. ([Bug#28358](#))
- Portability problems caused by use of `isinf()` were corrected. ([Bug#28240](#))
- Using a `TEXT` local variable in a stored routine in an expression such as `SET var = SUBSTRING(var, 3)` produced an incorrect result. ([Bug#27415](#))
- A large filesort could result in a division by zero error and a server crash. ([Bug#27119](#))
- The server could abort or deadlock for `INSERT DELAYED` statements for which another insert was performed implicitly (for example, via a stored function that inserted a row). ([Bug#21483](#))
- The server could hang for `INSERT IGNORE ... ON DUPLICATE KEY UPDATE` if an update failed. ([Bug#28000](#))
- Quoted labels in stored routines were mishandled, rendering the routines unusable. ([Bug#21513](#))
- Changes to some system variables should invalidate statements in the query cache, but invalidation did not happen. ([Bug#27792](#))
- Some `ALTER TABLE` statements that worked in MySQL 5.0 did not work in 5.1. ([Bug#28415](#))
- Flow control optimization in stored routines could cause exception handlers to never return or execute incorrect logic. ([Bug#26977](#))
- An attempt to execute `CREATE TABLE ... SELECT` when a temporary table with the same name already existed led to the insertion of data into the temporary table and creation of an empty non-temporary table. ([Bug#24508](#))
- Concurrent execution of `CREATE TABLE ... SELECT` and other statements involving the target table suffered from various race conditions, some of which might have led to deadlocks. ([Bug#24738](#))
- `CREATE TABLE IF NOT EXISTS ... SELECT` caused a server crash if the target table already existed and had a `BEFORE INSERT` trigger. ([Bug#20903](#))
- A statement of the form `CREATE TABLE IF NOT EXISTS t1 SELECT f1() AS i` failed with a deadlock error if the stored function `f1()` referred to a table with the same name as the to-be-created table. Now it correctly produces a message that the table already exists. ([Bug#22427](#))
- Deadlock occurred for attempts to execute `CREATE TABLE IF NOT EXISTS ... SELECT` when `LOCK TABLES` had been used to acquire a read lock on the target table. ([Bug#20662](#))
- It was not possible to use the value `-9223372036854775808` (that is, `-MAXVALUE + 1`) when specifying a `LIST` partition. ([Bug#28005](#))
- Some `InnoDB` variables were missing from the output of `mysqld --verbose --help`. ([Bug#26987](#))
- `CAST()` to `DECIMAL` did not check for overflow. ([Bug#27957](#))
- Views ignored precision for `CAST()` operations. ([Bug#27921](#))
- For `InnoDB`, in some rare cases the optimizer preferred a more expensive `ref` access to a less expensive range access. ([Bug#28189](#))
- A query with a `NOT IN` subquery predicate could cause a crash when the left operand of the predicate evaluated to `NULL`. ([Bug#28375](#))
- Comparisons of `DATE` or `DATETIME` values for the `IN()` function could yield incorrect results. ([Bug#28133](#))
- `LOAD DATA` did not use `CURRENT_TIMESTAMP` as the default value for a `TIMESTAMP` column for which no value was provided. ([Bug#27670](#))

C.1.7. Changes in MySQL 5.1.19 Carrier Grade Edition

This section contains change history information for MySQL Cluster 5.1 Carrier Grade Edition releases based on MySQL 5.1.19.

C.1.7.1. Changes in release MySQL 5.1.19-ndb-6.3.1-beta (04 July 2007)

This is a new Beta development release, fixing a recently discovered bug.

Like all releases for MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.19-ndb-6.3.1. The file `mysqlcom-5.1.19-ndb-6.3.1-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.19-ndb-6.3.0, as well as all bugfixes and feature changes which were added in the mainline 5.1.19 release; information about these can be found in [Section C.1.6, “Changes in release 5.1.19 \(25 May 2007\)”](#).

This release fixes the following bug:

- Batching of transactions was not handled correctly in some cases. ([Bug#29525](#))

C.1.7.2. Changes in release MySQL 5.1.19-ndb-6.3.0-beta (02 July 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.19-ndb-6.3.0. The file `mysqlcom-5.1.19-ndb-6.3.0-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.18-ndb-6.2.1, MySQL 5.1.18-ndb-6.2.2 (see [Section C.1.9, “Changes in MySQL 5.1.18 Carrier Grade Edition”](#)), and MySQL 5.1.19-ndb-6.2.3 (see [Section C.1.7.4, “Changes in release MySQL 5.1.19-ndb-6.2.3-beta \(02 July 2007\)”](#)), as well as all bugfixes and feature changes which were added in the mainline 5.1.19 release; information about these can be found in [Section C.1.6, “Changes in release 5.1.19 \(25 May 2007\)”](#).

Functionality added or changed:

- **MySQL Cluster Replication:** This release implements conflict resolution, which makes it possible to determine on a per-table basis whether or not an update to a given row on the master should be applied on the slave. For more information, see [Section 16.10.10, “MySQL Cluster Replication Conflict Resolution”](#).
- A new configuration parameter `ODirect` causes **NDB** to attempt using `O_DIRECT` writes for LCP, backups, and redo logs, often lowering CPU usage.

C.1.7.3. Changes in release MySQL 5.1.19-ndb-6.2.4-beta (04 July 2007)

This is a new Beta development release, fixing recently discovered bugs.

Like all releases for MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL](#)

[Cluster from Source Code](#)". You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.19-ndb-6.2.4. The file `mysqlcom-5.1.19-ndb-6.2.4-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.18-ndb-6.2.1, MySQL 5.1.18-ndb-6.2.3 (see [Section C.1.9, "Changes in MySQL 5.1.18 Carrier Grade Edition"](#)), and MySQL 5.1.19-ndb-6.2.4, as well as all bugfixes and feature changes which were added in the mainline 5.1.19 release; information about these can be found in [Section C.1.6, "Changes in release 5.1.19 \(25 May 2007\)"](#). Also included are most (but not all) bugfixes made in the MCCGE 6.1.x series through MySQL 5.1.15-ndb-6.1.16.

This release fixes the following bugs:

- When restarting a data node, queries could hang during that node's start phase 5, and only continue once the node entered phase 6. ([Bug#29364](#))
- Replica redo logs were inconsistently handled during a system restart. ([Bug#29354](#))
- Batching of transactions was not handled correctly in some cases. ([Bug#29525](#))
- **Disk Data:** Disk data meta-information that existed in `ndb` might not be visible to `mysqld`. ([Bug#28720](#))
- **Disk Data:** The number of free extents was incorrectly reported for some tablespaces. ([Bug#28642](#))
- **Disk Data:** Performing Disk Data schema operations during a node restart could cause forced shutdowns of other data nodes. ([Bug#29501](#))

C.1.7.4. Changes in release MySQL 5.1.19-ndb-6.2.3-beta (02 July 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, "MySQL Installation Using a Source Distribution"](#), and in [Section 16.4.1, "Building MySQL Cluster from Source Code"](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.19-ndb-6.2.3. The file `mysqlcom-5.1.19-ndb-6.2.3-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.18-ndb-6.2.1 and MySQL 5.1.18-ndb-6.2.2 (see [Section C.1.9, "Changes in MySQL 5.1.18 Carrier Grade Edition"](#)), as well as all bugfixes and feature changes which were added in the mainline 5.1.19 release; information about these can be found in [Section C.1.6, "Changes in release 5.1.19 \(25 May 2007\)"](#). Also included are most (but not all) bugfixes made in the MCCGE 6.1.x series through MySQL 5.1.15-ndb-6.1.16.

Functionality added or changed:

- Backup reporting functionality has been significantly enhanced in this release:
 - A new configuration parameter `BackupReportFrequency` now makes it possible to cause the management client to provide status reports at regular intervals as well as for such reports to be written to the cluster log (depending on cluster event logging levels). See [Section 16.4.4.5, "Defining Data Nodes"](#), for more information about this parameter.
 - A new `REPORT BackupStatus` command has been added in the cluster management client which allows you to obtain a backup status report at any time during a backup. For more about this command, see [Section 16.7.2, "Commands in the MySQL Cluster Management Client"](#).
 - `ndb_restore` now provides running reports of its progress when restoring a backup. In addition, a complete report status report on the backup is written to the cluster log.

- A new memory allocator has been implemented for the **NDB** kernel, which allocates memory to tables 32K page by 32K page rather than allocating it in variable-sized chunks as previously. This removes much of the memory overhead that was associated with the old memory allocator.
- A new **NdbRecord** object has been added to the **NDB** API. This object provides mapping to a record stored in **NDB**.
- Read ahead was implemented for backups of Disk Data tables, resulting in a 10 to 15% increase in backup speed of Disk Data tables. (Bug#29099)
- Batching of updates on cluster replication slaves, enabled using the `--slave-allow-batching` option for **mysqld**. See [Section 16.10.6, “Starting Replication \(Single Replication Channel\)”](#), for more information.
- A new configuration parameter **ODirect** causes **NDB** to attempt using **O_DIRECT** writes for LCP, backups, and redo logs, often lowering CPU usage.

This release fixes the following bugs:

- Setting **MaxNoOfTables** very low and relative to **DataMemory** caused **OUT OF MEMORY IN NDB KERNEL** errors when inserting relatively small amounts of data into **NDB** tables. (Bug#24173)
- The actual value of **MaxNoOfOpenFiles** as used by the cluster was offset by 1 from the value set in **config.ini**. This meant that setting **InitialNoOfOpenFiles** to the same value always caused an error. (Bug#28749)
- **Cluster Replication:** When replicating **MyISAM** or **InnoDB** tables to a MySQL Cluster, it was not possible to determine exactly what had been applied following a shutdown of the slave cluster or **mysqld** process. (Bug#26783)
- **Cluster APIs:** An invalid error code could be set on transaction objects by **BLOB** handling code. (Bug#28724)
- It is now possible to set the maximum size of the allocation unit for table memory using the **MaxAllocate** configuration parameter. (Bug#29044)
- **Cluster APIs:** The timeout set using the MGM API **ndb_mgm_set_timeout()** function was incorrectly interpreted as seconds rather than as milliseconds. (Bug#29063)
- **ndb_error_reporter** now preserves timestamps on files. (Bug#29074)
- Attempting to restore a **NULL** row to a **VARBINARY** column caused **ndb_restore** to fail. (Bug#29103)
- **auto_increment_increment** and **auto_increment_offset** are now supported for **NDB** tables. (Bug#26342)
- When shutting down **mysqld**, the **NDB** binlog process was not shut down before log cleanup began. (Bug#28949)
- The management client's response to **START BACKUP WAIT COMPLETED** did not include the backup ID. (Bug#27640)
- A query having a large **IN(...)** or **NOT IN(...)** list in the **WHERE** condition on an **NDB** table could cause **mysqld** to crash. (Bug#29185)
- **UPDATE IGNORE** statements involving the primary keys of multiple tables could result in data corruption. (Bug#28719)
- When a node failed to respond to a **COPY_GCI** signal as part of a global checkpoint, the master node was killed instead of the node that actually failed. (Bug#29331)
- If at least 2 files were involved in **REDO** invalidation, then file 0 of page 0 was not updated and so pointed to an invalid part of the redo log. (Bug#29057)
- The wrong data pages were sometimes invalidated following a global checkpoint. (Bug#29067)
- An invalid comparison made during **REDO** validation that could lead to an **ERROR WHILE READING REDO LOG** condition. (Bug#29118)
- **Disk Data:** When dropping a page, the stack's bottom entry could sometime be left “cold” rather than “hot”, violating the rules for stack pruning. (Bug#29176)

- Memory corruption could occur due to a problem in the `DBTUP` kernel block. (Bug#29229)
- The `NDB` transporter would hang when more than 1024 signals were received at one time. (Bug#28443)
- Having large amounts of memory locked caused swapping to disk. (Bug#28751)
- It was not possible to set a separate watchdog timeout at startup. (Bug#28899)
- LCP files were not removed following an initial system restart. (Bug#28726)
- **Disk Data:** Repeated `INSERT` and `DELETE` operations on a Disk Data table having one or more large `VARCHAR` columns could cause data nodes to fail. (Bug#20612)
- A corrupt schema file could cause a `FILE ALREADY OPEN` error. (Bug#28770)
- A race condition could occur between `NODE_FAILREP` and `COPY_GCIREQ` events. (Bug#28717)
- While loading data to the cluster after a version upgrade, the data nodes failed due to `ndbrequire` failures in `PGMAN`. (Bug#28525)
- A fast global checkpoint under high load with a high usage of the redo buffer caused data nodes to fail. (Bug#28653)
- The actual value of `MaxNoOfOpenFiles` as used by the cluster was offset by 1 from the value set in `config.ini`. This meant that setting `InitialNoOpenFiles` to the same value always caused an error. (Bug#28749)
- In the event that two data nodes in the same node group and participating in a GCP crashed before they had written their respective `P0.sysfile` files, `QMGR` could refuse to start, issuing an invalid `INSUFFICIENT NODES FOR RESTART` error instead. (Bug#29167)
- **Disk Data:** When dropping a page, the stack's bottom entry could sometime be left “cold” rather than “hot”, violating the rules for stack pruning. (Bug#29176)

C.1.8. Changes in release 5.1.18 (08 May 2007)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- **Incompatible change:** The `INFORMATION_SCHEMA.EVENTS` and `mysql.event` tables have been changed to facilitate replication of events. When upgrading to MySQL 5.1.18, you must run `mysql_upgrade` prior to working with events. Until you have done so, any statement relating to the Event Scheduler or these tables (including `SHOW EVENTS`) will fail with the errors `EXPECTED FIELD STATUS AT POSITION 12 TO HAVE TYPE ENUM ( 'ENABLED', 'SLAVESIDE_DISABLED', 'DISABLED' ), FOUND ENUM( 'ENABLED', 'DISABLED' )` and `TABLE MYSQL.EVENT IS DAMAGED. CAN NOT OPEN`.

These changes were made as part of fixes for the following bugs:

- The effects of scheduled events were not replicated (that is, binary logging of scheduled events did not work). (Bug#16421, Bug#17857)
- Effects of scheduled events on a replication master were both replicated and executed on the slave, causing double execution of events. (Bug#20384)
- `CREATE FUNCTION` statements and their effects were not replicated correctly. (Bug#17671)

For more information, see [Section 15.3.1.5, “Replication of Invoked Features”](#).

Important

When upgrading to MySQL 5.1.18 or later from a previous MySQL version and scheduled events have been used, the upgrade utilities do not accommodate changes in event-related system tables. As a workaround, you can dump events before the upgrade, then restore them from the dump afterwards. This issue was fixed in MySQL 5.1.20 (see [Bug#28521](#)).

- Prior to this release, when `DATE` values were compared with `DATETIME` values the time portion of the `DATETIME` value was ignored. Now a `DATE` value is coerced to the `DATETIME` type by adding the time portion as “00:00:00”. To mimic the old behavior use the `CAST()` function in the following way: `SELECT date_field = CAST(NOW() as DATE);`. ([Bug#28929](#))
- `mysqld_multi` now understands the `--no-defaults`, `--defaults-file`, and `--defaults-extra-file` options. The `--config-file` option is deprecated; if given, it is treated like `--defaults-extra-file`. ([Bug#27390](#))
- The output of `mysql --xml` and `mysqldump --xml` now includes a valid XML namespace. ([Bug#25946](#))
- The `mysql_create_system_tables` script was removed because `mysql_install_db` no longer uses it in MySQL 5.1.
- Renamed the `old_mode` system variable to `old`.
- If you use SSL for a client connection, you can tell the client not to authenticate the server certificate by specifying neither `--ssl-ca` nor `--ssl-capath`. The server still verifies the client according to any applicable requirements established via `GRANT` statements for the client, and it still uses any `--ssl-ca/--ssl-capath` values that were passed to server at startup time. ([Bug#25309](#))
- Added a `MASTER_SSL_VERIFY_SERVER_CERT` option for the `CHANGE MASTER` statement, and a `Master_SSL_Verify_Server_Cert` output column to the `SHOW SLAVE STATUS` statement. The option value also is written to the `master.info` file. ([Bug#19991](#))
- **NDB Cluster (APIs):** The MGM API now supports explicit setting of network timeouts using the `ndb_mgm_set_timeout()` function. A utility function `ndb_mgm_number_of_mgmd_in_connect_string()` is also implemented to facilitate calculation of timeouts based on the number of management servers in the cluster.

For more information, see `ndb_mgm_set_timeout()`, and `ndb_mgm_number_of_mgmd_in_connect_string()`.

- **NDB Cluster:** The behavior of the `ndb_restore` utility has been changed as follows:
 - It is now possible to restore selected databases or tables using `ndb_restore`. ([Bug#26899](#))
 - Several options have been added for use with `ndb_restore --print-data` to facilitate the creation of structured data dump files. These options can be used to make dumps made using `ndb_restore` more like those produced by `mysqldump`. ([Bug#26900](#))

For details of these changes, see [Section 16.8.3, “ndb_restore — Restore a Cluster Backup”](#).

- **NDB Cluster (Replication)/Incompatible Change:** The definition of the `mysql.ndb_apply_status` table has changed such that an online upgrade is not possible from MySQL 5.1.17 or earlier for a replication slave cluster; you must shut down all SQL nodes as part of the upgrade procedure. See [Section 16.5.2, “Cluster Upgrade and Downgrade Compatibility”](#) before upgrading for details.

For more information about the changes to `mysql.ndb_apply_status` see [Section 16.10.4, “Cluster Replication Schema and Tables”](#).

- **NDB Cluster (Replication):** Some circular replication setups are now supported for MySQL Cluster. See [Section 16.10.3, “Known Issues in MySQL Cluster Replication”](#), for detailed information. ([Bug#25688](#), [Bug#17095](#))
- **NDB Cluster:** The following changes were made for the `ndb_size.pl` utility:
 - When `ndb_size.pl` calculates a value for a given configuration parameter that is less than the default value, it now suggests the default value instead. ([Bug#24227](#))
 - The dependency on `HTML:Template` was removed. ([Bug#24228](#))
- Several additional data types are supported for columns in `INFORMATION_SCHEMA` tables: `DATE`, `TIME`, `BLOB`, `FLOAT`, and all integer types. ([Bug#27047](#))
- **NDB Cluster:** The internal specifications for columns in `NDB` tables has changed to allow compatibility with future MySQL

Cluster releases that are expected to implement online adding and dropping of columns. This change is not backwards compatible with earlier MySQL versions.

See [the related note](#) in [Section 16.5.2, “Cluster Upgrade and Downgrade Compatibility”](#), for important information prior to upgrading a MySQL Cluster to MySQL 5.1.18 or later from MySQL 5.1.17 or earlier.

See also [Bug#28205](#).

- If a set function *S* with an outer reference *S(outer_ref)* cannot be aggregated in the outer query against which the outer reference has been resolved, MySQL interprets *S(outer_ref)* the same way that it would interpret *S(const)*. However, standard SQL requires throwing an error in this situation. An error now is thrown for such queries if the [ANSI SQL](#) mode is enabled. ([Bug#27348](#))
- Added `--write-binlog` option for `mysqlcheck`. This option is enabled by default, but can be given as `--skip-write-binlog` to cause `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements generated by `mysqlcheck` not to be written to the binary log. ([Bug#26262](#))
- The plugin interface and its handling of system variables was changed. Command-line options such as `--skip-innodb` now cause an error if `InnoDB` is not built-in or plugin-loaded. You should use `--loose-skip-innodb` if you do not want any error even if `InnoDB` is not available. The `--loose` prefix modifier should be used for all command-line options where you are uncertain whether the plugin exists and when you want the operation to proceed even if the option is necessarily ignored due to the absence of the plugin. (For a description of how `--loose` works, see [Section 4.3.1, “Using Options on the Command Line”](#).)
- New command-line options: To alleviate ambiguities in variable names, all variables related to plugins can be specified using a `plugin` part in the name. For example, every time where we used to have `innodb` in the command-line options, you can now write `plugin-innodb`:

```
--skip-plugin-innodb
--plugin-innodb-buffer-pool-size=#
```

Furthermore, this is the preferred syntax. It helps to avoid ambiguities when a plugin, say, `wait`, has an option called `timeout`. `--wait-timeout` will still set a system variable, but `--plugin-wait-timeout` will set the plugin variable. Also, there is a new command-line option `--plugin-load` to install or load plugins at initialization time without using the `mysql.plugin` table.

- Storage engine plugins may now be uninstalled at run time. However, a plugin is not actually uninstalled until after its reference count drops to zero. The `default_storage_engine` system variable consumes a reference count, so uninstalling will not complete until said reference is removed.

Bugs fixed:

- **Security fix:** If a stored routine was declared using `SQL SECURITY INVOKER`, a user who invoked the routine could gain privileges. ([CVE-2007-2692](#), [Bug#27337](#))
- **Security fix:** The requirement of the `DROP` privilege for `RENAME TABLE` was not being enforced. ([CVE-2007-2691](#), [Bug#27515](#))
- **Security fix:** A user with only the `ALTER` privilege on a partitioned table could obtain information about the table that should require the `SELECT` privilege. ([CVE-2007-2693](#), [Bug#23675](#))
- The patches for [Bug#19370](#) and [Bug#21789](#) were reverted.
- **NDB Cluster** (Disk Data): Creating an excessive number of Disk Data tables (1000 or more) could cause data nodes to fail. ([Bug#24951](#))
- **NDB Cluster:** When a cluster node suffered a “hard” failure (such as a power failure or loss of a network connection) TCP sockets to the “vanished” node were maintained indefinitely. Now socket-based transporters check for a response and terminate the socket if there is no activity on the socket after 2 hours. ([Bug#24793](#))
- **NDB Cluster:** Under certain rare circumstances, `ndbd` could get caught in an infinite loop when one transaction took a read lock and then a second transaction attempted to obtain a write lock on the same tuple in the lock queue. ([Bug#28073](#))
- **NDB Cluster:** Under some circumstances, a node restart could fail to update the Global Checkpoint Index (GCI). ([Bug#28023](#))
- **NDB Cluster** (Disk Data): It was possible to drop the last remaining datafile in a tablespace (using an `ALTER TABLESPACE`

statement), even though there was still an empty table using the tablespace. ([Bug#21699](#))

It should be noted that the datafile could be not dropped if the table still contained any rows, so this bug involved no loss of data.

- **NDB Cluster:** `INSERT IGNORE` wrongly ignored `NULL` values in unique indexes. ([Bug#27980](#))
- **NDB Cluster:** The name of the month “March” was given incorrectly in the cluster error log. ([Bug#27926](#))
- **NDB Cluster** (Cluster Replication / Disk Data): An issue with replication of Disk Data tables could in some cases lead to node failure. ([Bug#28161](#))
- **NDB Cluster** (APIs): For `BLOB` reads on operations with lock mode `LM_CommittedRead`, the lock mode was not upgraded to `LM_Read` before the state of the `BLOB` had already been calculated. The **NDB** API methods affected by this problem included the following:
 - `NdbOperation::readTuple()`
 - `NdbScanOperation::readTuples()`
 - `NdbIndexScanOperation::readTuples()` ([Bug#27320](#))
- **NDB Cluster** (Disk Data): Setting the value of the `UNDO BUFFER SIZE` to 64K or less in a `CREATE LOGFILE GROUP` statement led to failure of cluster data nodes. ([Bug#24560](#))
- **NDB Cluster:** The cluster waited 30 seconds instead of 30 milliseconds before reading table statistics. ([Bug#28093](#))
- **NDB Cluster:** It was not possible to add a unique index to an **NDB** table while in single user mode. ([Bug#27710](#))
- **NDB Cluster:** Performing a delete followed by an insert during a local checkpoint could cause a `ROWID ALREADY ALLOCATED` error. ([Bug#27205](#))
- **NDB Cluster** (Disk Data): When restarting a data node following the creation of a large number (~200) of Disk Data objects, the cluster could not assign a node ID to the restarting node. ([Bug#25741](#))
- **NDB Cluster:** Adding of indexes online failed for **NDB** tables having `BLOB` or `TEXT` columns. ([Bug#25431](#))
- **NDB Cluster:** The `ndb_resize.pl` utility did not calculate memory usage for indexes correctly. ([Bug#24229](#))
- **NDB Cluster:** While a data node was stopped, dropping a table then creating an index on a different table caused that node to fail during restart. This was due to the re-use of the dropped table's internal ID for the index without verifying that the index now referred to a different database object. ([Bug#21755](#))
- **NDB Cluster** (Disk Data): When in single user mode, it was possible to create log file groups and tablespaces from any SQL node connected to the cluster. ([Bug#27712](#))
- **NDB Cluster** (Disk Data): Changing a column specification or issuing a `TRUNCATE` statement on a Disk Data table caused the table to become an in-memory table. ([Bug#24667](#), [Bug#25296](#))

This fix supersedes an incomplete fix that was made for this issue in MySQL 5.1.15.

- **NDB Cluster** (Replication): Setting `SQL_LOG_BIN` to zero did not disable binary logging. ([Bug#27076](#))

This issue affected only the **NDB** storage engine.

- **NDB Cluster** (Replication): It was possible for API nodes to begin interacting with the cluster subscription manager before they were fully connected to the cluster. ([Bug#27728](#))
- **NDB Cluster** (Replication): Under very high loads, checkpoints could be read or written with checkpoint indexes out of order. ([Bug#27651](#))
- **NDB Cluster:** **NDB** tables having `MEDIUMINT AUTO_INCREMENT` columns were not restored correctly by `ndb_restore`, causing spurious duplicate key errors. This issue did not affect `TINYINT`, `INT`, or `BIGINT` columns with `AUTO_INCREMENT`. ([Bug#27775](#))
- **NDB Cluster:** **NDB** tables with indexes whose names contained space characters were not restored correctly by `ndb_restore`

(the index names were truncated). (Bug#27758)

- **NDB Cluster** (Disk Data): Changes to a Disk Data table made as part of a transaction could not be seen by the client performing the changes until the transaction had been committed. (Bug#27757)
- **NDB Cluster**: An **INSERT** followed a delete **DELETE** on the same **NDB** table caused a memory leak. (Bug#27756)
(This bug was introduced by the fix for Bug#20612.)
- **NDB Cluster**: Some queries that updated multiple tables were not backed up or replicated correctly. (Bug#27748)
- **NDB Cluster**: Joins on multiple tables containing **BLOB** columns could cause data nodes run out of memory, and to crash with the error **NDBOBJECTIDMAP::EXPAND UNABLE TO EXPAND**. (Bug#26176)
- **NDB Cluster**: When trying to create an **NDB** table after the server was started with **--ndbcluster** but without **--ndb-connectstring**, **mysqld** produced a memory allocation error. (Bug#27359)
- **NDB Cluster** (APIs): Using **NdbBlob::writeData()** to write data in the middle of an existing blob value (that is, updating the value) could overwrite some data past the end of the data to be changed. (Bug#27018)
- **NDB Cluster** (Replication): Trying to replicate a large number of frequent updates with a relatively small relay log (**max-relay-log-size** set to 1M or less) could cause the slave to crash. (Bug#27529)
- **NDB Cluster** (Replication): An SQL node acting as a replication master server could be a single point of failure; that is, if it failed, the replication slave had no way of knowing this, which could result in a mismatch of data between the master and the slave. (Bug#21494)
- **NDB Cluster** (Disk Data): **CREATE TABLE ... LIKE disk_data_table** created an in-memory **NDB** table. (Bug#25875)
- **NDB Cluster** (Disk Data): Creating an excessive number of data files for a single tablespace caused data nodes to crash. (Bug#24521)
- **NDB Cluster**: Under certain rare circumstances, **DROP TABLE** or **TRUNCATE** of an **NDB** table could cause a node failure or forced cluster shutdown. (Bug#27581)
- **NDB Cluster**: Memory usage of a **mysqld** process grew even while idle. (Bug#27560)
- **NDB Cluster**: Using more than 16GB for **DataMemory** caused problems with variable-size columns. (Bug#27512)
- **NDB Cluster**: In an **NDB** table having a **TIMESTAMP** column using **DEFAULT CURRENT_TIMESTAMP**, that column would assume a random value when another column in the same row was updated. (Bug#27127)
- **NDB Cluster**: Performing **ALTER TABLE ... ENGINE=MERGE** on an **NDB** table caused **mysqld** to crash. (Bug#26898)
- **NDB Cluster**: The Cluster table handler did not set bits in null bytes correctly. (Bug#26591)
- **NDB Cluster**: In some cases, **AFTER UPDATE** and **AFTER DELETE** triggers on **NDB** tables that referenced subject table did not see the results of operation which caused invocation of the trigger, but rather saw the row as it was prior to the update or delete operation.

This was most noticeable when an update operation used a subquery to obtain the rows to be updated. An example would be **UPDATE tbl1 SET col2 = val1 WHERE tbl1.col1 IN (SELECT col3 FROM tbl2 WHERE c4 = val2)** where there was an **AFTER UPDATE** trigger on table **tbl1**. In such cases, the trigger would fail to execute.

The problem occurred because the actual update or delete operations were deferred to be able to perform them later as one batch. The fix for this bug solves the problem by disabling this optimization for a given update or delete if the table has an **AFTER** trigger defined for this operation. (Bug#26242)

- **NDB Cluster**: **START BACKUP NOWAIT** caused a spurious **OUT OF BACKUP RECORD** error in the management client (**START BACKUP** and **START BACKUP WAIT STARTED** performed normally). (Bug#25446)
- **NDB Cluster**: When trying to create tables on an SQL node not connected to the cluster, a misleading error message **TABLE 'TBL_NAME' ALREADY EXISTS** was generated. The error now generated is **COULD NOT CONNECT TO STORAGE ENGINE**. (Bug#18676)

- **NDB Cluster**: A data node failing while another data node was restarting could leave the cluster in an inconsistent state. In certain rare cases, this could lead to a race condition and the eventual forced shutdown of the cluster. ([Bug#27466](#))
- **NDB Cluster**: When using the **MemReportFrequency** configuration parameter to generate periodic reports of memory usage in the cluster log, **DataMemory** usage was not always reported for all data nodes. ([Bug#27444](#))
- **NDB Cluster**: Error messages displayed when running in single user mode were inconsistent. ([Bug#27021](#))
- **NDB Cluster**: On Solaris, the value of an **NDB** table column declared as **BIT (33)** was always displayed as **0**. ([Bug#26986](#))
- **NDB Cluster** (Replication): An **UPDATE** on the master became a **DELETE** on slaves. ([Bug#27378](#))
- During a call to **mysql_change_user()**, when authentication fails or the database to change to is unknown, a subsequent call to any function that does network communication leads to packets out of order. This problem was introduced in MySQL 5.1.14. ([Bug#25371](#))
- **ON DUPLICATE KEY UPDATE** failed for a table partitioned by **KEY** on a primary key **VARCHAR** column. ([Bug#27123](#))
- The fix for [Bug#17212](#) provided correct sort order for misordered output of certain queries, but caused significant overall query performance degradation. (Results were correct (good), but returned much more slowly (bad).) The fix also affected performance of queries for which results were correct. The performance degradation has been addressed. ([Bug#27531](#))
- On Windows, connection handlers did not properly decrement the server's thread count when exiting. ([Bug#25621](#))
- **SELECT COUNT(*)** from a table containing a **DATETIME NOT NULL** column could produce spurious warnings with the **NO_ZERO_DATE** SQL mode enabled. ([Bug#22824](#))
- Nested aggregate functions could be improperly evaluated. ([Bug#27363](#))
- Using **CAST()** to convert **DATETIME** values to numeric values did not work. ([Bug#23656](#))
- Early **NULL**-filtering optimization did not work for **eq_ref** table access. ([Bug#27939](#))
- Non-grouped columns were allowed by ***** in **ONLY_FULL_GROUP_BY** SQL mode. ([Bug#27874](#))
- Debug builds on Windows generated false alarms about uninitialized variables with some Visual Studio runtime libraries. ([Bug#27811](#))
- An error message suggested the use of **mysql_fix_privilege_tables** after an upgrade, but the recommended program is now **mysql_upgrade**. ([Bug#27818](#))
- **mysqld** did not check the length of option values and could crash with a buffer overflow for long values. ([Bug#27715](#))
- Index hints (**USE INDEX**, **IGNORE INDEX**, **FORCE INDEX**) cannot be used with **FULLTEXT** indexes, but were not being ignored. ([Bug#25951](#))
- **mysql_upgrade** did not detect failure of external commands that it runs. ([Bug#26639](#))
- **mysql_upgrade** did not pass a password to **mysqlcheck** if one was given. ([Bug#25452](#))
- On Windows, **mysql_upgrade** was sensitive to lettercase of the names of some required components. ([Bug#25405](#))
- The result set of a query that used **WITH ROLLUP** and **DISTINCT** could lack some rollup rows (rows with **NULL** values for grouping attributes) if the **GROUP BY** list contained constant expressions. ([Bug#24856](#))
- Some upgrade problems are detected and better error messages suggesting that **mysql_upgrade** be run are produced. ([Bug#24248](#))
- A performance degradation was observed for outer join queries to which a not-exists optimization was applied. ([Bug#28188](#))
- The **LEAST()** and **GREATEST()** functions compared **DATE** and **DATETIME** values as strings, which in some cases could lead to an incorrect result. ([Bug#27759](#))
- **SELECT * INTO OUTFILE ... FROM INFORMATION_SCHEMA.schemata** failed with an **Access denied** error, even for a user who has the **FILE** privilege. ([Bug#28181](#))

- Certain queries that used uncorrelated scalar subqueries caused `EXPLAIN` to crash. (Bug#27807)
- `INSERT...ON DUPLICATE KEY UPDATE` could cause `Error 1032: Can't find record in ...` for inserts into an InnoDB table unique index using key column prefixes with an underlying `utf8` string column. (Bug#13191)
- On Linux, the server could not create temporary tables if `lower_case_table_names` was set to 1 and the value of `tmpdir` was a directory name containing any uppercase letters. (Bug#27653)
- A slave that used `--master-ssl-cipher` could not connect to the master. (Bug#21611)
- `mysqldump` crashed if it got no data from `SHOW CREATE PROCEDURE` (for example, when trying to dump a routine defined by a different user and for which the current user had no privileges). Now it prints a comment to indicate the problem. It also returns an error, or continues if the `--force` option is given. (Bug#27293)
- Several math functions produced incorrect results for large unsigned values. `ROUND()` produced incorrect results or a crash for a large number-of-decimals argument. (Bug#24912)
- For storage engines that allow the current auto-increment value to be set, using `ALTER TABLE ... ENGINE` to convert a table from one such storage engine to another caused loss of the current value. (For storage engines that do not support setting the value, it cannot be retained anyway when changing the storage engine.) (Bug#25262)
- Comparison of a `DATE` with a `DATETIME` did not treat the `DATE` as having a time part of `00:00:00`. (Bug#27590)
- A multiple-table `UPDATE` could return an incorrect rows-matched value if, during insertion of rows into a temporary table, the table had to be converted from a `MEMORY` table to a `MyISAM` table. (Bug#22364)
- The omission of leading zeros in dates could lead to erroneous results when these were compared with the output of certain date and time functions. (Bug#16377)
- If `CREATE TABLE t1 LIKE t2` failed due to a full disk, an empty `t2.frm` file could be created but not removed. This file then caused subsequent attempts to create a table named `t2` to fail. This is easily corrected at the filesystem level by removing the `t2.frm` file manually, but now the server removes the file if the create operation does not complete successfully. (Bug#25761)
- The `MERGE` storage engine could return incorrect results when several index values that compare equality were present in an index (for example, `'gross'` and `'gross '`, which are considered equal but have different lengths). (Bug#24342)
- For InnoDB tables, a multiple-row `INSERT` of the form `INSERT INTO t (id...) VALUES (NULL...) ON DUPLICATE KEY UPDATE id=VALUES(id)`, where `id` is an `AUTO_INCREMENT` column, could cause `ERROR 1062 (23000): Duplicate entry...` errors or lost rows. (Bug#27650)
- `mysql_install_db` is supposed to detect existing system tables and create only those that do not exist. Instead, it was exiting with an error if tables already existed. (Bug#27783)
- Failure to allocate memory associated with `transaction_prealloc_size` could cause a server crash. (Bug#27322)
- Aborting a statement on the master that applied to a non-transactional statement broke replication. The statement was written to the binary log but not completely executed on the master. Slaves receiving the statement executed it completely, resulting in loss of data synchrony. Now an error code is written to the error log so that the slaves stop without executing the aborted statement. (That is, replication stops, but synchrony to the point of the stop is preserved and you can investigate the problem.) (Bug#26551)
- The `AUTO_INCREMENT` value would not be correctly reported for InnoDB tables when using `SHOW CREATE TABLE` statement or `mysqldump` command. (Bug#23313)
- Creating a temporary table with InnoDB when using the one-file-per-table setting, when the host filesystem for temporary tables is `tmpfs` would cause an assertion in `mysqld`. This was due to the use of `O_DIRECT` when opening the temporary table file. (Bug#26662)
- An interaction between `SHOW TABLE STATUS` and other concurrent statements that modify the table could result in a divide-by-zero error and a server crash. (Bug#27516)
- The `decimal.h` header file was incorrectly omitted from binary distributions. (Bug#27456)
- `mysqldump` could not connect using SSL. (Bug#27669)
- `yaSSL` crashed on pre-Pentium Intel CPUs. (Bug#21765)

- The test for the `MYSQL_OPT_SSL_VERIFY_SERVER_CERT` option for `mysql_options()` was performed incorrectly. Also changed as a result of this bugfix: The `arg` option for the `mysql_options()` C API function was changed from `char *` to `void *`. (Bug#24121)
- Comparisons using row constructors could fail for rows containing `NULL` values. (Bug#27704)
- Performing a `UNION` on two views that had had `ORDER BY` clauses resulted in an `Unknown column` error. (Bug#27786)
- The `CRC32()` function returns an unsigned integer, but the metadata was signed, which could cause certain queries to return incorrect results. (For example, queries that selected a `CRC32()` value and used that value in the `GROUP BY` clause.) (Bug#27530)
- A race condition between `DROP TABLE` and `SHOW TABLE STATUS` could cause the latter to display incorrect information. (Bug#27499)
- `mysqldump` would not dump a view for which the `DEFINER` no longer exists. (Bug#26817)
- Changing a `utf8` column in an `InnoDB` table to a shorter length did not shorten the data values. (Bug#20095)
- The server did not shut down cleanly. (Bug#27310)
- Using `SET GLOBAL` to change the `lc_time_names` system variable had no effect on new connections. (Bug#22648)
- The XML output representing an empty result was an empty string rather than an empty `<resultset/>` element. (Bug#27608)
- The range optimizer could consume a combinatorial amount of memory for certain classes of `WHERE` clauses. (Bug#26624)
- `mysqlbinlog` produced different output with the `-R` option than without it. (Bug#27171)
- A stored function invocation in the `WHERE` clause was treated as a constant. (Bug#27354)
- `mysqldump` could not dump the log tables. (Bug#26121)
- Implicit conversion of `9912101` to `DATE` did not match `CAST(9912101 AS DATE)`. (Bug#23093)
- Some equi-joins containing a `WHERE` clause that included a `NOT IN` subquery caused a server crash. (Bug#27870)
- A memory leak in the event scheduler that was uncovered by Valgrind was fixed. (Bug#27733)
- For queries that used `ORDER BY` with `InnoDB` tables, if the optimizer chose an index for accessing the table but found a covering index that enabled the `ORDER BY` to be skipped, no results were returned. (Bug#24778)
- Group relay log rotation updated only the log position and not the name, causing the slave to stop. (Bug#27583)
- Conversion of `DATETIME` values in numeric contexts sometimes did not produce a double (`YYYYMMDDHHMMSS.uuuuuuu`) value. (Bug#16546)
- Passing nested row expressions with different structures to an `IN` predicate caused a server crash. (Bug#27484)
- `SELECT DISTINCT` could return incorrect results if the select list contained duplicated columns. (Bug#27659)
- A subquery could get incorrect values for references to outer query columns when it contained aggregate functions that were aggregated in outer context. (Bug#27321)
- In some cases, the optimizer preferred a range or full index scan access method over lookup access methods when the latter were much cheaper. (Bug#19372)
- Duplicates were not properly identified among (potentially) long strings used as arguments for `GROUP_CONCAT(DISTINCT)`. (Bug#26815)
- For `InnoDB`, fixed consistent-read behavior of the first read statement, if the read was served from the query cache, for the `READ COMMITTED` isolation level. (Bug#21409)
- The binary log incompatibility introduced by the fix for Bug#22583 was corrected. (Bug#27779)
- Row-based replication of `MyISAM` to non-`MyISAM` tables did not work correctly for `BIT` columns. This has been corrected, but the fix introduces an incompatibility into the binary log format. (Bug#22583) (The incompatibility is corrected by the fix for

[Bug#27779](#).)

- Duplicate members in `SET` definitions were not detected. Now they result in a warning; if strict SQL mode is enabled, an error occurs instead. ([Bug#27069](#))
- If the name of a table given to `myisamchk -rq` was a packed table and the name included the `.MYI` extension, `myisamchk` incorrectly created a file with a `.MYI.MYI` extension. ([Bug#26782](#))
- For `INSERT INTO ... SELECT` where index searches used column prefixes, insert errors could occur when key value type conversion was done. ([Bug#26207](#))
- For `SHOW ENGINE INNODB STATUS`, the `LATEST DEADLOCK INFORMATION` was not always cleared properly. ([Bug#25494](#))
- On Windows, if the server was installed as a service, it did not auto-detect the location of the data directory. ([Bug#20376](#))
- The `FEDERATED` engine did not allow the local and remote tables to have different names. ([Bug#26257](#))
- The `NO_DIR_IN_CREATE` server SQL mode was not enforced for partitioned tables. ([Bug#24633](#))
- On Windows, trying to use backslash (`\`) characters in paths for `DATA DIRECTORY` and `INDEX DIRECTORY` when creating partitioned tables caused MySQL to crash. ([Bug#25141](#), [Bug#26074](#))

(You must use `/` characters when specifying paths for these options, regardless of platform. See [Section 17.1, “Overview of Partitioning in MySQL”](#), for an example using absolute paths for `DATA DIRECTORY` and `INDEX DIRECTORY` when creating a partitioned table on Windows.)
- A damaged or missing `mysql.event` table caused `SHOW VARIABLES` to fail. ([Bug#23631](#))
- Database and table names have a maximum length of 64 characters (even if they contain multi-byte characters), but were being truncated to 64 bytes. ([Bug#21432](#))
- If a rotate event occurred in the middle of a non-transaction group, the group position would be updated by the rotate event indicating an illegal group start position that was effectively inside a group. This can happen if, for example, a rotate occurs between an `In-tvar` event and the associated `Query` event, or between the table map events and the rows events when using row-based replication. ([Bug#23171](#))
- `mysqldump` could crash or exhibit incorrect behavior when some options were given very long values, such as `-fields-terminated-by="some very long string"`. The code has been cleaned up to remove a number of fixed-sized buffers and to be more careful about error conditions in memory allocation. ([Bug#26346](#))
- Setting a column to `NOT NULL` with an `ON DELETE SET NULL` clause foreign key crashes the server. ([Bug#25927](#))
- The values displayed for the `Innodb_row_lock_time`, `Innodb_row_lock_time_avg`, and `Innodb_row_lock_time_max` status variables were incorrect. ([Bug#23666](#))
- `COUNT(decimal_expr)` sometimes generated a spurious truncation warning. ([Bug#21976](#))
- With `NO_AUTO_VALUE_ON_ZERO` SQL mode enabled, `LOAD DATA` operations could assign incorrect `AUTO_INCREMENT` values. ([Bug#27586](#))
- Incorrect results could be returned for some queries that contained a select list expression with `IN` or `BETWEEN` together with an `ORDER BY` or `GROUP BY` on the same expression using `NOT IN` or `NOT BETWEEN`. ([Bug#27532](#))
- For the `INFORMATION_SCHEMA.SESSION_STATUS` and `GLOBAL_STATUS` tables, some status values were incorrectly converted to the data type of the `VARIABLE_VALUE` column. ([Bug#27327](#))
- Queries containing subqueries with `COUNT(*)` aggregated in an outer context returned incorrect results. This happened only if the subquery did not contain any references to outer columns. ([Bug#27257](#))
- Use of an aggregate function from an outer context as an argument to `GROUP_CONCAT()` caused a server crash. ([Bug#27229](#))
- Restoration of the default database after stored routine or trigger execution on a slave could cause replication to stop if the database no longer existed. ([Bug#25082](#))
- String truncation upon insertion into an integer or year column did not generate a warning (or an error in strict mode). ([Bug#26359](#),

[Bug#27176](#))

- In out-of-memory conditions, the server might crash or otherwise not report an error to the Windows event log. ([Bug#27490](#))
- The temporary file-creation code was cleaned up on Windows to improve server stability. ([Bug#26233](#))
- Out-of-memory errors were not reported. Now they are written to the error log. ([Bug#26844](#))
- `mysqldump` crashed for `MERGE` tables if the `--complete-insert (-c)` option was given. ([Bug#25993](#))
- Corrupted `MyISAM` tables that have different definitions in the `.frm` and `.MYI` tables might cause a server crash. ([Bug#25908](#))
- In certain situations, `MATCH ... AGAINST` returned false hits for `NULL` values produced by `LEFT JOIN` when no full-text index was available. ([Bug#25729](#))
- `OPTIMIZE TABLE` might fail on Windows when it attempts to rename a temporary file to the original name if the original file had been opened, resulting in loss of the `.MYD` file. ([Bug#25521](#))
- `GRANT` statements were not replicated if the server was started with the `--replicate-ignore-table` or `--replicate-wild-ignore-table` option. ([Bug#25482](#))
- `MBROverlaps()` returned incorrect values in some cases. ([Bug#24563](#))
- A problem in handling of aggregate functions in subqueries caused predicates containing aggregate functions to be ignored during query execution. ([Bug#24484](#))
- Improved out-of-memory detection when sending logs from a master server to slaves, and log a message when allocation fails. ([Bug#26837](#))
- `MBRDisjoint()`, `MBRequal()`, `MBRIntersects()`, `MBROverlaps()`, `MBRTouches()`, and `MBRWithin()` were inadvertently omitted from recent versions of MySQL (5.1.14 to 5.1.17). ([Bug#24563](#))
- `SHOW CREATE VIEW` qualified references to stored functions in the view definition with the function's database name, even when the database was the default database. This affected `mysqldump` (which uses `SHOW CREATE VIEW` to dump views) because the resulting dump file could not be used to reload the database into a different database. `SHOW CREATE VIEW` now suppresses the database name for references to functions in the default database. ([Bug#23491](#))
- When MySQL logged slow query information to a `CSV` table, it used an incorrect formula to calculate the `query_time` and `lock_time` values. ([Bug#27638](#))
- With `innodb_file_per_table` enabled, attempting to rename an `InnoDB` table to a non-existent database caused the server to exit. ([Bug#27381](#))
- `mysql_install_db` could terminate with an error after failing to determine that a system table already existed. ([Bug#27022](#))
- For `InnoDB` tables having a clustered index that began with a `CHAR` or `VARCHAR` column, deleting a record and then inserting another before the deleted record was purged could result in table corruption. ([Bug#26835](#))
- `make_win_bin_dist` neglected to copy some required `MyISAM` table files. ([Bug#26922](#))
- Fixed a possible buffer overflow in `SHOW PROCEDURE CODE`. ([Bug#26303](#))
- Selecting the result of `AVG()` within a `UNION` could produce incorrect values. ([Bug#24791](#))
- Access via `my_pread()` or `my_pwrite()` to table files larger than 2GB could fail on some systems. ([Bug#24566](#))
- An `INTO OUTFILE` clause is allowed only for the final `SELECT` of a `UNION`, but this restriction was not being enforced correctly. ([Bug#23345](#))
- Duplicate entries were not assessed correctly in a `MEMORY` table with a `BTREE` primary key on a `utf8 ENUM` column. ([Bug#24985](#))
- For `MyISAM` tables, `COUNT(*)` could return an incorrect value if the `WHERE` clause compared an indexed `TEXT` column to the empty string (`' '`). This happened if the column contained empty strings and also strings starting with control characters such as tab or newline. ([Bug#26231](#))
- For `DELETE FROM tbl_name ORDER BY col_name` (with no `WHERE` or `LIMIT` clause), the server did not check whether

`col_name` was a valid column in the table. ([Bug#26186](#))

- `ALTER VIEW` requires the `CREATE VIEW` and `DROP` privileges for the view. However, if the view was created by another user, the server erroneously required the `SUPER` privilege. ([Bug#26813](#))
- `mysqlbinlog --base64-output` produced invalid SQL. ([Bug#26194](#))
- In a view, a column that was defined using a `GEOMETRY` function was treated as having the `LONGBLOB` data type rather than the `GEOMETRY` type. ([Bug#27300](#))
- Some views could not be created even when the user had the requisite privileges. ([Bug#24040](#))
- With the `NO_AUTO_VALUE_ON_ZERO` SQL mode enabled, `LAST_INSERT_ID()` could return 0 after `INSERT ... ON DUPLICATE KEY UPDATE`. Additionally, the next rows inserted (by the same `INSERT`, or the following `INSERT` with or without `ON DUPLICATE KEY UPDATE`), would insert 0 for the auto-generated value if the value for the `AUTO_INCREMENT` column was `NULL` or missing. ([Bug#23233](#))
- Executing an `INSERT ... SELECT ... FROM INFORMATION_SCHEMA.GLOBAL_STATUS` statement from within an event caused a server crash. ([Bug#26174](#))
- Having the `EXECUTE` privilege for a routine in a database should make it possible to `USE` that database, but the server returned an error instead. This has been corrected. As a result of the change, `SHOW TABLES` for a database in which you have only the `EXECUTE` privilege returns an empty set rather than an error. ([Bug#9504](#))
- When `RAND()` was called multiple times inside a stored procedure, the server did not write the correct random seed values to the binary log, resulting in incorrect replication. ([Bug#25543](#))
- `SOUNDEX()` returned an invalid string for international characters in multi-byte character sets. ([Bug#22638](#))
- Row equalities in `WHERE` clauses could cause memory corruption. ([Bug#27154](#))
- `GROUP BY` on a `ucs2` column caused a server crash when there was at least one empty string in the column. ([Bug#27079](#))
- Evaluation of an `IN()` predicate containing a decimal-valued argument caused a server crash. ([CVE-2007-2583](#)) ([Bug#27362](#), [Bug#27513](#))
- Storing `NULL` values in spatial fields caused excessive memory allocation and crashes on some systems. ([Bug#27164](#))
- `mysql_stmt_fetch()` did an invalid memory deallocation when used with the embedded server. ([Bug#25492](#))
- For `FEDERATED` tables, `SHOW CREATE TABLE` could fail when the table name was longer than the connection name. ([Bug#27036](#))
- In a `MEMORY` table, using a `BTREE` index to scan for updatable rows could lead to an infinite loop. ([Bug#26996](#))
- The range optimizer could cause the server to run out of memory. ([Bug#26625](#))
- `CREATE SERVER`, `DROP SERVER`, and `ALTER SERVER` did not require any privileges. Now these statements require the `SUPER` privilege. ([Bug#25671](#))
- Concurrent `CREATE SERVER` and `ALTER SERVER` statements could cause a deadlock. ([Bug#25721](#))
- Difficult repair or optimization operations could cause an assertion failure, resulting in a server crash. ([Bug#25289](#))

C.1.9. Changes in MySQL 5.1.18 Carrier Grade Edition

This section contains change history information for MySQL Cluster 5.1 Carrier Grade Edition releases based on MySQL 5.1.18.

C.1.9.1. Changes in release MySQL 5.1.18-ndb-6.2.2-beta (07 May 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using

the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.18-ndb-6.2.2. The file `mysqlcom-5.1.18-ndb-6.2.2-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.16-ndb-6.2.0 and MySQL 5.1.18-ndb-6.2.1, which includes all bugfixes and feature changes which were added in the mainline 5.1.18 release; information about these can be found in [Section C.1.8, “Changes in release 5.1.18 \(08 May 2007\)”](#).

Functionality added or changed:

- Added a new `mysqld` option `--ndb-cluster-connection-pool` that allows a single MySQL server to use multiple connections to the cluster. This allows for scaling out using multiple MySQL clients per SQL node instead of or in addition to using multiple SQL nodes with the cluster.

For more information about this option, see the description in [Section 5.2.2, “Command Options”](#).

- New cluster management client `DUMP` commands were added to aid in tracking transactions, scan operations, and locks. See `DUMP 2350`, `DUMP 2352`, and `DUMP 2550`, for more information.

C.1.9.2. Changes in release MySQL 5.1.18-ndb-6.2.1-beta (Released 30 April 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Warning

This release was withdrawn after production and should no longer be used.

Like all releases for MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.16-ndb-6.2.0. The file `mysqlcom-5.1.16-ndb-6.2.0-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.18-ndb-6.2.0, which includes all bugfixes and feature changes which were added in the mainline 5.1.18 release; information about these can be found in [Section C.1.8, “Changes in release 5.1.18 \(08 May 2007\)”](#).

Important

Upgrading to MySQL 5.1.18-ndb-6.2.1 from a previous release. The internal specifications for columns in `NDB` tables changed to allow compatibility with future MySQL Cluster and MySQL Cluster 5.1 Carrier Grade Edition releases that are expected to implement online adding and dropping of columns. This change is not backwards compatible with MySQL 5.1.16-ndb-6.2.0, ndb-6.1.x versions prior to MySQL 5.1.15-ndb-6.1.7, or MySQL Cluster mainline releases prior to 5.1.18.

See the related note in [Section 16.5.2, “Cluster Upgrade and Downgrade Compatibility”](#), for important information prior to upgrading a MySQL Cluster to MySQL 5.1.15-ndb-6.1.7 or later from MySQL 5.1.15-ndb-6.1.6 or an earlier ndb-6.1.x release.

See also [Bug#28205](#).

This release fixes the following bugs:

- Multiple operations involving deletes followed by reads were not handled correctly. (Bug#28276)

Note

This issue could also affect MySQL Cluster Replication.

- **Cluster APIs:** Using `NdbBlob::writeData()` to write data in the middle of an existing blob value (that is, updating the value) could overwrite some data past the end of the data to be changed. (Bug#27018)
- **Cluster Replication:** Trying to replicate a large number of frequent updates with a relatively small relay log (`max-re-lay-log-size` set to 1M or less) could cause the slave to crash. (Bug#27529)
- **Cluster Replication:** An SQL node acting as a replication master server could be a single point of failure; that is, if it failed, the replication slave had no way of knowing this, which could result in a mismatch of data between the master and the slave. (Bug#21494)
- Memory usage of a `mysqld` process grew even while idle. (Bug#27560)
- The Cluster table handler did not set bits in null bytes correctly. (Bug#26591)
- When trying to create tables on an SQL node not connected to the cluster, a misleading error message `TABLE 'TBL_NAME' ALREADY EXISTS` was generated. The error now generated is `COULD NOT CONNECT TO STORAGE ENGINE`. (Bug#18676)
- Incorrect handling of fragmentation in a node takeover during a restart could cause stale data to be copied to the starting node, leading eventually to failure of the node. (Bug#27434)
- Incorrect handling of fragmentation in a node takeover during a restart could cause stale data to be copied to the starting node, leading eventually to failure of the node. (Bug#27434)
- After putting the cluster in single user mode from one MySQL server, trying to drop an NDB table from a second MySQL server also connected to the cluster would cause the second MySQL server to hang. (Bug#27254)
- `mysqld` could crash shortly after a data node failure following certain DML operations. (Bug#27169)
- An incorrect assertion was made when sending a `TCKEYFAILREF` or `TCKEYCONF` message to a failed data node. (Bug#26814)

C.1.10. Changes in release 5.1.17 (04 April 2007)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- **Incompatible change:** Scheduled events now use the MySQL server time zone to determine their schedules, rather than UTC as in previous releases. Because of this change, scheduled event metadata now includes time zone information, which can be seen in the `TIME_ZONE` column of the `INFORMATION_SCHEMA.EVENTS` table and the `Time zone` column in the output of the `SHOW EVENTS` statement. These columns have been added in this release, along with a `time_zone` column in the `mysql.event` table. Due to these changes, events created in previous versions of MySQL cannot be created, viewed, or used until `mysql.event` has been upgraded. (Bug#16420)
- **Important change:** The `CREATE EVENT` and `ALTER EVENT` statements now support a `DEFINER` clause, similar to that used in the `CREATE TRIGGER` statement. (Bug#16425)

See [Section 21.2.2, “CREATE EVENT Syntax”](#), for detailed information.

- **Important change:** The following options for controlling replication master configuration on a slave are now deprecated.

- `--master-host`
- `--master-user`
- `--master-password`
- `--master-port`
- `--master-connect-retry`
- `--master-ssl`
- `--master-ssl-ca`
- `--master-ssl-capath`
- `--master-ssl-cert`
- `--master-ssl-cipher`
- `--master-ssl-key`

To change the master configuration on a slave you should use the `CHANGE MASTER` statement. (See also #21490)

- Statements that affect `mysql` database tables now are written to the binary log using the following rules:
 - Data manipulation statements such as `INSERT` that change data in `mysql` database tables directly are logged according to the settings of the `binlog_format` system variable.
 - Statements such as `GRANT` that change the `mysql` database indirectly are logged as statements regardless of the value of `binlog_format`.

For more details, see [Section 5.11.4.4, “Logging Format for Changes to `mysql` Database Tables”](#). (Bug#25091)

- Prepared statements now use the query cache under the conditions described in [Section 6.5.4.1, “How the Query Cache Operates”](#). (Bug#735)
- Added the `--secure-file-priv` option for `mysqld`, which limits the effect of the `LOAD_FILE()` function and the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements to work only with files in a given directory. (Bug#18628)
- Added the `--service-startup-timeout` option for `mysql.server` to specify how long to wait for the server to start. If the server does not start within the timeout period, `mysql.server` exits with an error. (Bug#26952)
- Added the `innodb_stats_on_metadata` system variable to enable control over whether `InnoDB` performs statistics gathering when metadata statements are executed. See [Section 13.5.4, “InnoDB Startup Options and System Variables”](#). (Bug#26598)
- Added the `hostname` system variable, which the server sets at startup to the server hostname.
- The syntax for index hints has been extended to enable more fine-grained control over the optimizer's selection of an execution plan for various phases of query processing. See [Section 12.2.7.2, “Index Hint Syntax”](#). (Bug#21174)
- Added the `thread_handling` system variable to control whether the server use a single thread or one thread per connection. The `--one-thread` option now is deprecated; use `--thread_handling=one-thread` instead.
- Added the `old_mode` system variable to cause the server to revert to certain behaviors present in older versions. Currently, this variable affects handling of index hints. See [Section 12.2.7.2, “Index Hint Syntax”](#).
- **NDB Cluster:** Added the `--skip-table-check` option (short form `-s`) for `ndb_restore`, which causes the restoration process to ignore any changes that may have occurred in table schemas after the backup was made. Previously, this was the default behavior. (Bug#24363)

See [Section 16.8.3, “`ndb\_restore` — Restore a Cluster Backup”](#), for more information.

- The server now includes a timestamp in error messages that are logged as a result of unhandled signals (such as `mysqld got`

signal 11 messages). ([Bug#24878](#))

- Prefix lengths for columns in [SPATIAL](#) indexes no longer can be specified. For tables created in older versions of MySQL that have [SPATIAL](#) indexes containing prefixed columns, dumping and reloading the table causes the indexes to be created with no prefixes. (The full column width of each column is indexed.) ([Bug#26794](#))
- Added a `--no-beep` option to `mysqladmin`. It suppresses the warning beep that is emitted by default for errors such as a failure to connect to the server. ([Bug#26964](#))

Bugs fixed:

- **Important note:** The parser accepted invalid code in SQL condition handlers, leading to server crashes or unexpected execution behavior in stored programs. Specifically, the parser allowed a condition handler to refer to labels for blocks that enclose the handler declaration. This was incorrect because block label scope does not include the code for handlers declared within the labeled block.

The parser now rejects this invalid construct, but if you upgrade in place (without dumping and reloading your databases), existing handlers that contain the construct still are invalid *even if they appear to function as you expect* and should be rewritten.

To find affected handlers, use `mysqldump` to dump all stored functions and procedures, triggers, and events. Then attempt to reload them into an upgraded server. Handlers that contain illegal label references will be rejected.

For more information about condition handlers and writing them to avoid invalid jumps, see [Section 19.2.8.2, “DECLARE Handlers”](#). ([Bug#26503](#))

- **Incompatible change:** [INSERT DELAYED](#) statements are not supported for [MERGE](#) tables, but the [MERGE](#) storage engine was not rejecting such statements, resulting in table corruption. Applications previously using [INSERT DELAYED](#) into [MERGE](#) table will break when upgrading to versions with this fix. To avoid the problem, remove [DELAYED](#) from such statements. ([Bug#26464](#))
- Creating a table on one SQL node while in single user mode caused other SQL nodes to crash. ([Bug#26997](#))
- [NDB Cluster](#): The output from `ndb_restore --print_data` was incorrect for a backup made of a database containing tables with [TINYINT](#) or [SMALLINT](#) columns. ([Bug#26740](#))
- [NDB Cluster](#): After putting the cluster in single user mode from one MySQL server, trying to drop an NDB table from a second MySQL server also connected to the cluster would cause the second MySQL server to hang. ([Bug#27254](#))
- [NDB Cluster](#) (Cluster APIs): You can now use the `ndb_mgm_check_connection()` function to determine whether a management server is running. See `ndb_mgm_check_connection()`.
- [NDB Cluster](#): `mysqld` could crash shortly after a data node failure following certain DML operations. ([Bug#27169](#))
- [NDB Cluster](#): The management client command `node_id STATUS` displayed the message `Node node_id: not connected` when `node_id` was not the node ID of a data node. ([Bug#21715](#))

Note

The [ALL STATUS](#) command in the cluster management client still displays status information for data nodes only. This is by design. See [Section 16.7.2, “Commands in the MySQL Cluster Management Client”](#), for more information.

- [NDB Cluster](#) (Cluster APIs): [NAND](#) and [NOR](#) operations with `NdbScanFilter` did not perform correctly. ([Bug#24568](#))
- [NDB Cluster](#) (Cluster Replication): The simultaneous failure of a data node and an SQL node could cause replication to fail. ([Bug#27005](#))
- [NDB Cluster](#) (Disk Data): Under some circumstances, a data node could fail during restart while flushing Disk Data UNDO logs. ([Bug#27102](#))
- [NDB Cluster](#) (Disk Data): Creating multiple Disk Data tables using different tablespaces could sometimes cause the cluster to fail. ([Bug#25992](#))
- [NDB Cluster](#) (Disk Data): `ALTER TABLE ... ADD COLUMN ...` on a Disk Data table moved data for existing non-indexed columns from the tablespace into memory. ([Bug#25880](#))
- [NDB Cluster](#) (Disk Data): `DROP INDEX` on a Disk Data table did not always move data from memory into the tablespace.

[Bug#25877](#))

- **NDB Cluster** (Disk Data): When creating a log file group, setting `INITIAL_SIZE` to less than `UNDO_BUFFER_SIZE` caused data nodes to crash. ([Bug#25743](#))
- **NDB Cluster**: It was not possible to set `LockPagesInMainMemory` equal to 0. ([Bug#27291](#))
- **NDB Cluster**: A race condition could sometimes occur if the node acting as master failed while node IDs were still being allocated during startup. ([Bug#27286](#))
- **NDB Cluster**: When a data node was taking over as the master node, a race condition could sometimes occur as the node was assuming responsibility for handling of global checkpoints. ([Bug#27283](#))
- **NDB Cluster**: A delete operation using a scan following by an insert using a scan could cause a data node to fail. ([Bug#27203](#))
- **NDB Cluster**: The same failed request from an API node could be handled by the cluster multiple times, resulting in reduced performance. ([Bug#27087](#))
- **NDB Cluster**: The failure of a data node while restarting could cause other data nodes to hang or crash. ([Bug#27003](#))
- **NDB Cluster**: `mysqld` processes would sometimes crash under high load. ([Bug#26825](#))
- **NDB Cluster**: When performing an upgrade or downgrade, no specific error information was made available when trying to upgrade data nodes or SQL nodes before upgrading management nodes. ([Bug#21296](#))
- **NDB Cluster**: Some values of `MaxNoOfTables` caused the error `JOB BUFFER CONGESTION` to occur. ([Bug#19378](#))
- **NDB Cluster**: An invalid pointer was returned following a `FSCLOSECONF` signal when accessing the REDO logs during a node restart or system restart. ([Bug#26515](#))
- **NDB Cluster** (Disk Data): A memory overflow could occur with tables having a large amount of data stored on disk, or with queries using a very high degree of parallelism on Disk Data tables. ([Bug#26514](#))
- **NDB Cluster** (Disk Data): Use of a tablespace whose `INITIAL_SIZE` was greater than 1 GB could cause the cluster to crash. ([Bug#26487](#))
- **NDB Cluster**: Using only the `--print_data` option (and no other options) with `ndb_restore` caused `ndb_restore` to fail. ([Bug#26741](#))

This bug was introduced by the fix for [Bug#14612](#).

- **NDB Cluster**: An infinite loop in an internal logging function could cause trace logs to fill up with `UNKNOWN SIGNAL TYPE` error messages and thus grow to unreasonable sizes. ([Bug#26720](#))
- While preparing prepared statements, the server acquired unnecessary table write locks. ([Bug#18326](#))
- For some operations, system tables in the `mysql` database must be accessed. For example, the `HELP` statement requires the contents of the server-side help tables, and `CONVERT_TZ()` might need to read the time zone tables. However, to perform such operations while a `LOCK TABLES` statement is in effect, the server required you to also lock the requisite system tables explicitly or a lock error occurred:

```
mysql> LOCK TABLE t1 READ;
Query OK, 0 rows affected (0.02 sec)
mysql> HELP HELP;
ERROR 1100 (HY000) at line 4: Table 'help_topic' was not
locked with LOCK TABLES
```

Now, the server implicitly locks the system tables for reading as necessary so that you need not lock them explicitly. These tables are treated as just described:

```
mysql.help_category
mysql.help_keyword
mysql.help_relation
mysql.help_topic
mysql.proc
mysql.time_zone
mysql.time_zone_leap_second
mysql.time_zone_name
mysql.time_zone_transition
```

mysql.time_zone_transition_type

If you want to explicitly place a **WRITE** lock on any of those tables with a **LOCK TABLES** statement, the table must be the only one locked; no other table can be locked with the same statement. (Bug#9953)

- For a stored procedure containing a **SELECT** statement that used a complicated join with an **ON** expression, the expression could be ignored during re-execution of the procedure, yielding an incorrect result. (Bug#20492)
- On Windows, debug builds of **mysqld** could fail with heap assertions. (Bug#25765)
- On Windows, debug builds of **mysqlbinlog** could fail with a memory error. (Bug#23736)
- **SELECT ... INTO OUTFILE** with a long **FIELDS ENCLOSED BY** value could crash the server. (Bug#27231)
- The server could hang during binary log rotation. (Bug#26079)
- Increasing the width of a **DECIMAL** column could cause column values to be changed. (Bug#24558)
- **DOUBLE** values such as **20070202191048.000000** were being treated as illegal arguments by **WEEK()**. (Bug#23616)
- Replication between master and slave would infinitely retry binary log transmission where the **max_allowed_packet** on the master was larger than that on the slave if the size of the transfer was between these two values. (Bug#23775)
- Setting **event_scheduler=1** or **event_scheduler=ON** caused the server to crash if the server had been started with **--skip-grant-tables**. Starting the server with **--skip-grant-tables** now causes **event_scheduler** to be set to **DISABLED** automatically, overriding any other value that may have been set. (Bug#26807)
- **SHOW CREATE EVENT** failed to display the **STARTS** and **ENDS** clauses for an event defined with **STARTS NOW()**, **ENDS NOW()**, or both. (Bug#26429)
- Invalid optimization of pushdown conditions for queries where an outer join was guaranteed to read only one row from the outer table led to results with too few rows. (Bug#26963)
- For **INSERT ... ON DUPLICATE KEY UPDATE** statements on tables containing **AUTO_INCREMENT** columns, **LAST_INSERT_ID()** was reset to 0 if no rows were successfully inserted or changed. “Not changed” includes the case where a row was updated to its current values, but in that case, **LAST_INSERT_ID()** should not be reset to 0. Now **LAST_INSERT_ID()** is reset to 0 only if no rows were successfully inserted or touched, whether or not touched rows were changed. (Bug#27033)

This bug was introduced by the fix for Bug#19978.

- An **INSERT ... ON DUPLICATE KEY UPDATE** statement might modify values in a table but not flush affected data from the query cache, causing subsequent selects to return stale results. This made the combination of query cache plus **ON DUPLICATE KEY UPDATE** very unreliable. (Bug#27006, Bug#27210)

This bug was introduced by the fix for Bug#19978.

- For **INSERT ... ON DUPLICATE KEY UPDATE** statements where some **AUTO_INCREMENT** values were generated automatically for inserts and some rows were updated, one auto-generated value was lost per updated row, leading to faster exhaustion of the range of the **AUTO_INCREMENT** column. (Bug#24432)

Because the original problem can affect replication (different values on master and slave), it is recommended that the master and its slaves be upgraded to the current version.

- For an **INSERT** statement that should fail due to a column with no default value not being assigned a value, the statement succeeded with no error if the column was assigned a value in an **ON DUPLICATE KEY UPDATE** clause, even if that clause was not used. (Bug#26261)
- A result set column formed by concatenation of string literals was incomplete when the column was produced by a subquery in the **FROM** clause. (Bug#26738)
- When using the result of **SEC_TO_TIME()** for time value greater than 24 hours in an **ORDER BY** clause, either directly or through a column alias, the rows were sorted incorrectly as strings. (Bug#26672)
- If the server was started with **--skip-grant-tables**, Selecting from **INFORMATION_SCHEMA** tables causes a server crash.

[Bug#26285](#))

- Adding **ORDER BY** clause to a query on an **InnoDB** table could cause the statement to return no result if the optimizer chose a covering index that enabled it to skip the **ORDER BY**. ([Bug#24778](#))
- **IN ((subquery))**, **IN (((subquery)))**, and so forth, are equivalent to **IN (subquery)**, which is always interpreted as a table subquery (so that it is allowed to return more than one row). MySQL was treating the “over-parenthesized” subquery as a single-row subquery and rejecting it if it returned more than one row. This bug primarily affected automatically generated code (such as queries generated by Hibernate), because humans rarely write the over-parenthesized forms. ([Bug#21904](#))
- For **MERGE** tables defined on underlying tables that contained a short **VARCHAR** column (shorter than four characters), using **ALTER TABLE** on at least one but not all of the underlying tables caused the table definitions to be considered different from that of the **MERGE** table, even if the **ALTER TABLE** did not change the definition. ([Bug#26881](#))
- If a thread previously serviced a connection that was killed, excessive memory and CPU use by the thread occurred if it later serviced a connection that had to wait for a table lock. ([Bug#25966](#))
- **CURDATE()** is less than **NOW()**, either when comparing **CURDATE()** directly (**CURDATE() < NOW()** is true) or when casting **CURDATE()** to **DATE** (**CAST(CURDATE() AS DATE) < NOW()** is true). However, storing **CURDATE()** in a **DATE** column and comparing **col_name < NOW()** incorrectly yielded false. This is fixed by comparing a **DATE** column as **DATETIME** for comparisons to a **DATETIME** constant. ([Bug#21103](#))
- A view on a join is insertable for **INSERT** statements that store values into only one table of the join. However, inserts were being rejected if the inserted-into table was used in a self-join because MySQL incorrectly was considering the insert to modify multiple tables of the view. ([Bug#25122](#))
- Expressions involving **SUM()**, when used in an **ORDER BY** clause, could lead to out-of-order results. ([Bug#25376](#))
- **LOAD DATA INFILE** sent an okay to the client before writing the binary log and committing the changes to the table had finished, thus violating ACID requirements. ([Bug#26050](#))
- Views that used a scalar correlated subquery returned incorrect results. ([Bug#26560](#))
- **IF(expr, unsigned_expr, unsigned_expr)** was evaluated to a signed result, not unsigned. This has been corrected. The fix also affects constructs of the form **IS [NOT] {TRUE|FALSE}**, which were transformed internally into **IF()** expressions that evaluated to a signed result. ([Bug#24532](#))

For existing views that were defined using **IS [NOT] {TRUE|FALSE}** constructs, there is a related implication. The definitions of such views were stored using the **IF()** expression, not the original construct. This is manifest in that **SHOW CREATE VIEW** shows the transformed **IF()** expression, not the original one. Existing views will evaluate correctly after the fix, but if you want **SHOW CREATE VIEW** to display the original construct, you must drop the view and re-create it using its original definition. New views will retain the construct in their definition.

- **BENCHMARK()** did not work correctly for expressions that produced a **DECIMAL** result. ([Bug#26093](#))
- For **MEMORY** tables, extending the length of a **VARCHAR** column with **ALTER TABLE** might result in an unusable table. ([Bug#26080](#))
- For some values of the position argument, the **INSERT()** function could insert a NUL byte into the result. ([Bug#26281](#))
- Creating a table with latin characters in the name caused the output of **SHOW FULL TABLES** to have **ERROR** for the table type. ([Bug#25081](#))
- Inserting **utf8** data into a **TEXT** column that used a single-byte character set could result in spurious warnings about truncated data. ([Bug#25815](#))
- **EXPLAIN EXTENDED** did not show **WHERE** conditions that were optimized away. ([Bug#22331](#))
- **INSERT DELAYED** statements inserted incorrect values into **BIT** columns. ([Bug#26238](#))
- For **expr IN(value_list)**, the result could be incorrect if **BIGINT UNSIGNED** values were used for **expr** or in the value list. ([Bug#19342](#))
- When a **TIME_FORMAT()** expression was used as a column in a **GROUP BY** clause, the expression result was truncated. ([Bug#20293](#))

- For `SUBSTRING()` evaluation using a temporary table, when `SUBSTRING()` was used on a `LONGTEXT` column, the `max_length` metadata value of the result was incorrectly calculated and set to 0. Consequently, an empty string was returned instead of the correct result. (Bug#15757)
- Use of a `GROUP BY` clause that referred to a stored function result together with `WITH ROLLUP` caused incorrect results. (Bug#25373)
- Use of a subquery containing `GROUP BY` and `WITH ROLLUP` caused a server crash. (Bug#26830)
- Use of a subquery containing a `UNION` with an invalid `ORDER BY` clause caused a server crash. (Bug#26661)
- In certain cases it could happen that deleting a row corrupted an `RTREE` index. This affected indexes on spatial columns. (Bug#25673)
- SSL connections failed on Windows. (Bug#26678)
- Added support for `--debugger=dbx` for `mysql-test-run.pl` and fixed support for `--debugger=devenv`, `--debugger=DevEnv`, and `--debugger=/path/to/devenv`. (Bug#26792)
- `X() IS NULL` and `Y() IS NULL` comparisons failed when `X()` and `Y()` returned `NULL`. (Bug#26038)
- `UNHEX() IS NULL` comparisons failed when `UNHEX()` returned `NULL`. (Bug#26537)
- The `REPEAT()` function did not allow a column name as the `count` parameter. (Bug#25197)
- On 64-bit Windows, large timestamp values could be handled incorrectly. (Bug#26536)
- In some error messages, inconsistent format specifiers were used for the translations in different languages. `comp_err` (the error message compiler) now checks for mismatches. (Bug#26571)
- On Windows, the server exhibited a file-handle leak after reaching the limit on the number of open file descriptors. (Bug#25222)
- A reference to a non-existent column in the `ORDER BY` clause of an `UPDATE ... ORDER BY` statement could cause a server crash. (Bug#25126)
- `mysqlimport` used a variable of the wrong type for the `--use-threads` option, which could cause a crash on some architectures. (Bug#23814)
- A multiple-row delayed insert with an auto increment column could cause duplicate entries to be created on the slave in a replication environment. (Bug#25507, Bug#26116)
- Using `mysqlbinlog` on a binary log would crash if there were a large number of row-based events related to a single statement. (Bug#25628)
- Duplicating the usage of a user variable in a stored procedure or trigger would not be replicated correctly to the slave. (Bug#25167)
- User defined variables used within stored procedures and triggers are not replicated correctly when operating in statement-based replication mode. (Bug#20141, Bug#14914)
- Loading data using `LOAD DATA INFILE` may not replicate correctly (due to character set incompatibilities) if the `character_set_database` variable is set before the data is loaded. (Bug#15126)
- `DROP TRIGGER` statements would not be filtered on the slave when using the `replication-wild-do-table` option. (Bug#24478)
- `SHOW ENGINE MUTEX STATUS` failed to produce an `UNKNOWN TABLE ENGINE` error. (Bug#24392)
See [Section 12.5.4.13, “SHOW ENGINE Syntax”](#).
- MySQL would not compile when configured using `--without-query-cache`. (Bug#25075)
- When using certain server SQL modes, the `mysql.proc` table was not created by `mysql_install_db`. In addition, the creation of this and other MySQL system tables was not checked for by `mysql-test-run.pl`. (Bug#23669, Bug#20166)
- It was not possible to use XPath keywords as tag names for expressions used in the `ExtractValue()` function. (Bug#24747)

- **VIEW** restrictions were applied to **SELECT** statements after a **CREATE VIEW** statement failed, as though the **CREATE** had succeeded. (Bug#25897)
- An **INSERT** trigger invoking a stored routine that inserted into a table other than the one on which the trigger was defined would fail with a **TABLE '...' DOESN'T EXIST** referring to the second table when attempting to delete records from the first table. (Bug#21825)
- A stored procedure that made use of cursors failed when the procedure was invoked from a stored function. (Bug#25345)
- When nesting stored procedures within a trigger on a table, a false dependency error was thrown when one of the nested procedures contained a **DROP TABLE** statement. (Bug#22580)
- When attempting to call a stored procedure creating a table from a trigger on a table **tbl** in a database **db**, the trigger failed with **ERROR 1146 (42S02): TABLE 'DB.TBL' DOESN'T EXIST**. However, the actual reason that such a trigger fails is due to the fact that **CREATE TABLE** causes an implicit **COMMIT**, and so a trigger cannot invoke a stored routine containing this statement. A trigger which does so now fails with **ERROR 1422 (HY000): EXPLICIT OR IMPLICIT COMMIT IS NOT ALLOWED IN STORED FUNCTION OR TRIGGER**, which makes clear the reason for the trigger's failure. (Bug#18914)
- Local variables in stored routines or triggers, when declared as the **BIT** type, were interpreted as strings. (Bug#12976)
- When a stored routine attempted to execute a statement accessing a nonexistent table, the error was not caught by the routine's exception handler. (Bug#8407, Bug#20713)
- **NOW()** returned the wrong value in statements executed at server startup with the **--init-file** option. (Bug#23240)
- Instance Manager did not remove the angel PID file on a clean shutdown. (Bug#22511)
- Setting the **slow_query_log_file** system variable caused log output to go to the general log, not the slow query log. (Bug#23225)
- The server could crash if two or more threads initiated query cache resize operation at moments very close in time. (Bug#23527)
- The conditions checked by the optimizer to allow use of indexes in **IN** predicate calculations were unnecessarily tight and were relaxed. (Bug#20420)
- Several deficiencies in resolution of column names for **INSERT ... SELECT** statements were corrected. (Bug#25831)
- Indexes on **TEXT** columns were ignored when **ref** accesses were evaluated. (Bug#25971)
- The update columns for **INSERT ... SELECT ... ON DUPLICATE KEY UPDATE** could be assigned incorrect values if a temporary table was used to evaluate the **SELECT**. (Bug#16630)
- A user-defined variable could be assigned an incorrect value if a temporary table was employed in obtaining the result of the query used to determine its value. (Bug#24010)
- Queries that used a temporary table for the outer query when evaluating a correlated subquery could return incorrect results. (Bug#23800)
- For index reads, the **BLACKHOLE** engine did not return end-of-file (which it must because **BLACKHOLE** tables contain no rows), causing some queries to crash. (Bug#19717)
- A query of type **index_merge**, and with a **WHERE** clause having the form **WHERE indexed_column_1=value_1 OR indexed_column_2=value_2** on a partitioned table caused the server to crash. (Bug#26117)

C.1.11. Changes in release 5.1.16 (26 February 2007)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

■ Note

After release, a trigger failure problem was found to have been introduced. (Bug#27006) Users affected by this issue should upgrade to MySQL 5.1.17, which corrects the problem.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- **Incompatible change:** `TRUNCATE TABLE` now requires the `DROP` privilege rather than the `DELETE` privilege. (Bug#23556).
- **NDB Cluster** (Cluster APIs) / **Incompatible change:** The `AbortOption` type is now a member of the `NdbOperation` class; its values and behavior have also changed. `NdbTransaction::AbortOption` can no longer be used, and applications written against the NDB API may need to be rewritten and recompiled to accomodate these changes. For more information about this change, see [The NdbOperation::AbortOption Type](#).

This also affects the behavior of the `NdbTransaction::execute()` method, which now reports failure only if the transaction was actually aborted. See [NdbTransaction::execute\(\)](#), for more information.

- **NDB Cluster:** A new configuration parameter `MemReportFrequency` allows for additional control of data node memory usage. Previously, only warnings at predetermined percentages of memory allocation were given; setting this parameter allows for that behavior to be overridden. For more information, see [Section 16.4.4.5, “Defining Data Nodes”](#).
- **NDB Cluster** (Cluster APIs): A new `ndb_mgm_get_cluster_log_loglevel()` function was added to the MGM API. For more information, see [ndb_mgm_get_cluster_log_loglevel\(\)](#).
- **NDB Cluster:** Previously, when a data node failed more than 8 times in succession to start, this caused a forced shutdown of the cluster. Now, when a data node fails to start 7 consecutive times, the node will not start again until it is started with the `-initial` option, and a warning to this effect is written to the error log. (Bug#25984)
- **NDB Cluster:** A number of new and more descriptive error messages covering transporter errors were added. (Bug#22025)
- **NDB Cluster:** In the event that all cluster management and API nodes are configured with `ArbitrationRank=0`, `ndb_mgmd` now issues the following warning when starting: `CLUSTER CONFIGURATION WARNING: NEITHER MGM NOR API NODES ARE CONFIGURED WITH ARBITRATOR, MAY CAUSE COMPLETE CLUSTER SHUTDOWN IN CASE OF HOST FAILURE.` (Bug#23546)
- The `localhost` anonymous user account created during MySQL installation on Windows now has no global privileges. Formerly this account had all global privileges. For operations that require global privileges, the `root` account can be used instead. (Bug#24496)

- In the `INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS` table, the `UNIQUE_CONSTRAINT_NAME` column incorrectly named the referenced table. Now it names the referenced constraint, and a new column, `REFERENCED_TABLE_NAME`, names the referenced table. (Bug#21713)
- `RAND()` now allows non-constant initializers (such as a column name) as its argument. In this case, the seed is initialized with the value for each invocation of `RAND()`. (One implication of this is that for equal argument values, `RAND()` will return the same value each time.) (Bug#6172)
- The bundled yaSSL library was upgraded to version 1.5.8.
- `CONNECTION` is no longer treated as a reserved word. (Bug#12204)

Bugs fixed:

- **Security fix:** Using an `INFORMATION_SCHEMA` table with `ORDER BY` in a subquery could cause a server crash. (CVE-2007-1420, Bug#24630, Bug#26556) We would like to thank Oren Isacson from Flowgate Security Consulting as well as Stefan Streichsbier from SEC Consult for informing us about this problem.
- **NDB Cluster:** Condition pushdown did not work with prepared statements. (Bug#26225)
- **NDB Cluster** (APIs): After defining a delete operation using (`NdbOperation::deleteTuple()`) on a nonexistent primary

key of a table having a **BLOB** or **TEXT** column, invoking `NdbTransaction::execute()` caused the calling application to enter an endless loop rather than raising an error. (Bug#24028)

- **NDB Cluster**: An inadvertent use of unaligned data caused `ndb_restore` to fail on some 64-bit platforms, including Sparc and Itanium-2. (Bug#26739)
- **NDB Cluster**: The **InvalidUndoBufferSize** error used the same error code (763) as the **IncompatibleVersions** error. **InvalidUndoBufferSize** now uses its own error code (779). (Bug#26490)
- **NDB Cluster**: Takeover for local checkpointing due to multiple failures of master nodes was sometimes incorrectly handled. (Bug#26457)
- **NDB Cluster**: The `LockPagesInMemory` parameter was not read until after distributed communication had already started between cluster nodes. When the value of this parameter was 1, this could sometimes result in data node failure due to missed heartbeats. (Bug#26454)
- **NDB Cluster**: Under some circumstances, following the restart of a management, all cluster data nodes would connect to it normally, but some of them subsequently failed to log any events to the management node. (Bug#26293)
- **NDB Cluster**: An error was produced when `SHOW TABLE STATUS` was used on an **NDB** table that had no **AUTO_INCREMENT** column. (Bug#21033)

Note

This improves on and supersedes an earlier fix that was made for this issue in MySQL 5.1.12.

- **NDB Cluster**: When a node failed due to there being insufficient disk space to perform a local checkpoint, there was no indication that this was the source of the problem. Such a condition now produces an appropriate error message. (Bug#20121)
- **NDB Cluster** (Cluster APIs): `libndbclient.so` was not versioned. (Bug#13522)
- **NDB Cluster**: The `ndb_size.tmpl` file (necessary for using the `ndb_size.pl` script) was missing from binary distributions. (Bug#24191)
- **NDB Cluster** (Cluster APIs / Disk Data): A delete and a read performed in the same operation could cause one or more of the cluster's data nodes to crash. This could occur when the operation affected more than 5 columns concurrently, or when one or more of the columns was of the **VARCHAR** type and was stored on disk. (Bug#25794)
- **NDB Cluster** (Replication): The error message `LAST_ERRNO: 4294967295, ERROR IN WRITE_ROWS EVENT` now supplies a valid error code. (Bug#19896)
- **NDB Cluster** (Replication): Under some circumstances, the binlog thread could shut down while the slave SQL thread was still using it. (Bug#26015, Bug#26019)
- **NDB Cluster**: A query with an **IN** clause against an **NDB** table employing explicit user-defined partitioning did not always return all matching rows. (Bug#25821)
- A memory leak could cause problems during a node or cluster shutdown or failure. (Bug#25997)
- **NDB Cluster**: An appropriate error message was not provided when there was insufficient REDO log file space for the cluster to start. (Bug#25801)
- **NDB Cluster**: An **UPDATE** using an **IN** clause on an **NDB** table on which there was a trigger caused `mysqld` to crash. (Bug#25522)
- **NDB Cluster**: A memory allocation failure in the cluster Subscription Manager could cause the cluster to crash. (Bug#25239)
- **NDB Cluster**: In the event that cluster backup parameters such as `BackupWriteSize` were incorrectly set, no appropriate error was issued to indicate that this was the case. (Bug#19146)
- **OPTIMIZE TABLE** caused a race condition in the I/O cache. (Bug#19978)
- If the duplicate key value was present in the table, `INSERT ... ON DUPLICATE KEY UPDATE` reported a row count indicating that a record was updated, even when no record actually changed due to the old and new values being the same. Now it reports a row count of zero. (Bug#19978)

- Some `UPDATE` statements were slower than in previous versions when the search key could not be converted to a valid value for the type of the search column. (Bug#24035)
- The `WITH CHECK OPTION` clause for views was ignored for updates of multiple-table views when the updates could not be performed on fly and the rows to update had to be put into temporary tables first. (Bug#25931)
- Using `ORDER BY` or `GROUP BY` could yield different results when selecting from a view and selecting from the underlying table. (Bug#26209)
- Storing values specified as hexadecimal values 64 or more bits long into `BIT(64)`, `BIGINT`, or `BIGINT UNSIGNED` columns did not raise any warning or error if the value was out of range. (Bug#22533)
- Inserting `DEFAULT` into a column with no default value could result in garbage in the column. Now the same result occurs as when inserting `NULL` into a `NOT NULL` column. (Bug#20691)
- The presence of `ORDER BY` in a view definition prevented the `MERGE` algorithm from being used to resolve the view even if nothing else in the definition required the `TEMPTABLE` algorithm. (Bug#12122)
- `ISNULL( DATE( NULL ) )` and `ISNULL( CAST( NULL AS DATE ) )` erroneously returned false. (Bug#23938)
- If a slave server closed its relay log (for example, due to an error during log rotation), the I/O thread did not recognize this and still tried to write to the log, causing a server crash. (Bug#10798)
- Collation for `LEFT JOIN` comparisons could be evaluated incorrectly, leading to improper query results. (Bug#26017)
- For the `IF( )` and `COALESCE( )` function and `CASE` expressions, large unsigned integer values could be mishandled and result in warnings. (Bug#22026)
- The number of `setsockopt( )` calls performed for reads and writes to the network socket was reduced to decrease system call overhead. (Bug#22943)
- A `WHERE` clause that used `BETWEEN` for `DATETIME` values could be treated differently for a `SELECT` and a view defined as that `SELECT`. (Bug#26124)
- `ORDER BY` on `DOUBLE` values could change the set of rows returned by a query. (Bug#19690)
- `LOAD DATA INFILE` did not work with pipes. (Bug#25807)
- `DISTINCT` queries that were executed using a loose scan for an `InnoDB` table that had been emptied caused a server crash. (Bug#26159)
- `ALTER TABLE` caused loss of `CASCADE` clauses for `InnoDB` tables. (Bug#24741)
- Type conversion errors during formation of index search conditions were not correctly checked, leading to incorrect query results. (Bug#22344)
- Within a stored routine, accessing a declared routine variable with `PROCEDURE ANALYSE( )` caused a server crash. (Bug#23782)
- Use of already freed memory caused SSL connections to hang forever. (Bug#19209)
- `mysql.server stop` timed out too quickly (35 seconds) waiting for the server to exit. Now it waits up to 15 minutes, to ensure that the server exits. (Bug#25341)
- A yaSSL program named `test` was installed, causing conflicts with the `test` system utility. It is no longer installed. (Bug#25417)
- `perror` crashed on some platforms due to failure to handle a `NULL` pointer. (Bug#25344)
- `mysql_kill( )` caused a server crash when used on an SSL connection. (Bug#25203)
- The `readline` library wrote to uninitialized memory, causing `mysql` to crash. (Bug#19474)
- yaSSL was sensitive to the presence of whitespace at the ends of lines in PEM-encoded certificates, causing a server crash. (Bug#25189)
- The `SEC_TO_TIME( )` and `QUARTER( )` functions sometimes did not handle `NULL` values correctly. (Bug#25643)

- The optimizer used a filesort rather than a `const` table read in some cases when the latter was possible. (Bug#16590)
- With `ONLY_FULL_GROUP_BY` enabled, the server was too strict: Some expressions involving only aggregate values were rejected as non-aggregate (for example, `MAX(a) - MIN(a)`). (Bug#23417)
- Indexes disabled with `ALTER TABLE ... DISABLE KEYS` could in some cases be used by specifying `FORCE INDEX`. (Bug#20604)
- The arguments of the `ENCODE()` and the `DECODE()` functions were not printed correctly, causing problems in the output of `EXPLAIN EXTENDED` and in view definitions. (Bug#23409)
- An error in the name resolution of nested `JOIN ... USING` constructs was corrected. (Bug#25575)
- A return value of `-1` from user-defined handlers was not handled well and could result in conflicts with server code. (Bug#24987)
- The server might fail to use an appropriate index for `DELETE` when `ORDER BY`, `LIMIT`, and a non-restricting `WHERE` are present. (Bug#17711)
- Use of `ON DUPLICATE KEY UPDATE` defeated the usual restriction against inserting into a join-based view unless only one of the underlying tables is used. (Bug#25123)
- View definitions that used the `!` operator were treated as containing the `NOT` operator, which has a different precedence and can produce different results. (Bug#25580).
- Some queries against `INFORMATION_SCHEMA` that used subqueries failed. (Bug#23299).
- For a `UNIQUE` index containing many `NULL` values, the optimizer would prefer the index for `col IS NULL` conditions over other more selective indexes. (Bug#25407).
- `GROUP BY` and `DISTINCT` did not group `NULL` values for columns that have a `UNIQUE` index. (Bug#25551).
- `ALTER TABLE ... ENABLE KEYS` acquired a global lock, preventing concurrent execution of other statements that use tables. (Bug#25044).
- For an `InnoDB` table with any `ON DELETE` trigger, `TRUNCATE TABLE` mapped to `DELETE` and activated triggers. Now a fast truncation occurs and triggers are not activated. (Bug#23556).
- For `ALTER TABLE`, using `ORDER BY expression` could cause a server crash. Now the `ORDER BY` clause allows only column names to be specified as sort criteria (which was the only documented syntax, anyway). (Bug#24562)
- `readline` detection did not work correctly on NetBSD. (Bug#23293)
- The `--with-readline` option for `configure` did not work for commercial source packages, but no error message was printed to that effect. Now a message is printed. (Bug#25530)
- If an `ORDER BY` or `GROUP BY` list included a constant expression being optimized away and, at the same time, containing single-row subselects that return more than one row, no error was reported. If a query requires sorting by expressions containing single-row subselects that return more than one row, execution of the query may cause a server crash. (Bug#24653)
- To enable installation of MySQL RPMs on Linux systems running RHEL 4 (which includes SE-Linux) additional information was provided to specify some actions that are allowed to the MySQL binaries. (Bug#12676)
- Queries that evaluate `NULL IN (SELECT ... UNION SELECT ...)` could produce an incorrect result (`FALSE` instead of `NULL`). (Bug#24085)
- Within stored routines or prepared statements, inconsistent results occurred with multiple use of `INSERT ... SELECT ... ON DUPLICATE KEY UPDATE` when the `ON DUPLICATE KEY UPDATE` clause erroneously tried to assign a value to a column mentioned only in its `SELECT` part. (Bug#24491)
- Expressions of the form `(a, b) IN (SELECT a, MIN(b) FROM t GROUP BY a)` could produce incorrect results when column `a` of table `t` contained `NULL` values while column `b` did not. (Bug#24420)
- Expressions of the form `(a, b) IN (SELECT c, d ...)` could produce incorrect results if `a`, `b`, or both were `NULL`. (Bug#24127)
- An `AFTER UPDATE` trigger on an `InnoDB` table with a composite primary key caused the server to crash. (Bug#25398)

- A query that contained an `EXIST` subquery with a `UNION` over correlated and uncorrelated `SELECT` queries could cause the server to crash. (Bug#25219)
- A query with `ORDER BY` and `GROUP BY` clauses where the `ORDER BY` clause had more elements than the `GROUP BY` clause caused a memory overrun leading to a crash of the server. (Bug#25172)
- Certain joins using `Range checked for each record` in the query execution plan could cause the server to crash. (Bug#24776)
- If a prepared statement accessed a view, access to the tables listed in the query after that view was checked in the security context of the view. (Bug#24404)
- A nested query on a partitioned table returned fewer records than on the corresponding non-partitioned table, when the subquery affected more than one partition. (Bug#24186)
- Passing a `NULL` value to a user-defined function from within a stored procedure crashes the server. (Bug#25382)

C.1.12. Changes in MySQL 5.1.16 Carrier Grade Edition

This section contains change history information for MySQL Cluster 5.1 Carrier Grade Edition releases based on MySQL 5.1.16.

C.1.12.1. Changes in release MySQL 5.1.16-ndb-6.2.0-beta (03 March 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at [ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.16-ndb-6.2.0](http://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.16-ndb-6.2.0). The file `mysqlcom-5.1.16-ndb-6.2.0-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes which were added in MySQL 5.1.15-ndb-6.1.1 as well as in the mainline 5.1.16 release; information about these can be found in [Section C.1.14.19, “Changes in release MySQL 5.1.15-ndb-6.1.1-beta \(01 February 2007\)”](#), and in [Section C.1.11, “Changes in release 5.1.16 \(26 February 2007\)”](#).

Important

Upgrading to MySQL 5.1.16-ndb-6.2.0 from a previous release. This release is not binary compatible with previous MySQL MySQL Cluster 5.1 Carrier Grade Edition or mainline MySQL 5.1 releases. This means that:

- You cannot perform an online upgrade to this release from any of the MySQL MySQL Cluster 5.1 Carrier Grade Edition releases based on MySQL 5.1.14 or MySQL 5.1.15, or from any of the mainline MySQL 5.1 releases. When upgrading from one of these versions, you must shut down the cluster, replace all binaries, then restart the cluster.
- You must recompile all NDB API and MGM API applications used with a previous version of MySQL Cluster, including those compiled against any of the MySQL MySQL Cluster 5.1 Carrier Grade Edition releases based on MySQL 5.1.14 or MySQL 5.1.15.

Functionality added or changed:

- **Cluster APIs.** The `Ndb::startTransaction()` method now provides an alternative interface for starting a transaction. See [Ndb::startTransaction\(\)](#), for more information.
- **Cluster APIs.** New methods have been added to the `Ndb_cluster_connection` class to facilitate iterating over existing `Ndb` objects. See [ndb_cluster_connection::get_next_ndb_object\(\)](#), for more information.

- A new `TcpBind_INADDR_ANY` configuration parameter allows data nodes node to bind `INADDR_ANY` instead of a hostname or IP address in the `config.ini` file.
- A new `--ndb-wait-connected` option has been added for `mysqld`, which causes `mysqld` to wait for the specified amount of time to connect to the cluster before starting to accept MySQL client connections.
- Memory allocation has been improved on 32-bit architectures that enables using close to 3GB for `DataMemory` and `IndexMemory` combined.
- It is now possible to disable arbitration by setting `ArbitrationRank` equal to `0` on all nodes.

No new bugs were fixed in this release.

C.1.13. Changes in release 5.1.15 (25 January 2007)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- **Incompatible change:** The following conditions apply to enabling the `read_only` system variable:
 - If you attempt to enable `read_only` while you have any explicit locks (acquired with `LOCK TABLES` or have a pending transaction, an error will occur.
 - If other clients hold explicit table locks or have pending transactions, the attempt to enable `read_only` blocks until the locks are released and the transactions end. While the attempt to enable `read_only` is pending, requests by other clients for table locks or to begin transactions also block until `read_only` has been set.
 - `read_only` can be enabled while you hold a global read lock (acquired with `FLUSH TABLES WITH READ LOCK`) because that does not involve table locks.

Previously, the attempt to enable `read_only` would return immediately even if explicit locks or transactions were pending, so some data changes could occur for statements executing in the server at the same time. ([Bug#11733](#), [Bug#22009](#))

- **Incompatible change:** Previously, the `DATE_FORMAT()` function returned a binary string. Now it returns a string with a character set and collation given by `character_set_connection` and `collation_connection` so that it can return month and weekday names containing non-ASCII characters. ([Bug#22646](#))

Important

When using `MERGE` tables the definition of the `MERGE` table and the `MyISAM` tables are checked each time the tables are opened for access (including any `SELECT` or `INSERT` statement. Each table is compared for column order, types, sizes and associated. If there is a difference in any one of the tables then the statement will fail.

Important

Previously, duplicate-key errors were indicated by the `ER_DUP_ENTRY` error code (1062). This code is no longer used. Instead, the server returns `ER_DUP_ENTRY_WITH_KEY_NAME` (1582), and the error message indicates the name of the index for which the duplicate occurred. Applications that test for duplicate keys should look for both error codes if they need to be compatible with current and older servers. (This change was reverted in MySQL 5.1.20; see [Bug#28842](#).)

- On Unix, when searching the standard locations for option files, MySQL programs now also look for `/etc/mysql/my.cnf` after checking for `/etc/my.cnf` and before checking the remaining locations. (Bug#25104)
- Remote servers for use with the `FEDERATED` storage engine now can be managed with the new `CREATE/ALTER/DROP SERVER` syntax.
- The `--skip-thread-priority` option now is enabled by default for binary Mac OS X distributions. Use of thread priorities degrades performance on Mac OS X. (Bug#18526)
- Added the `--disable-grant-options` option to `configure`. If `configure` is run with this option, the `--bootstrap`, `--skip-grant-tables`, and `--init-file` options for `mysqld` are disabled and cannot be used. For Windows, the `configure.js` script recognizes the `DISABLE_GRANT_OPTIONS` flag, which has the same effect.
- Partitioning of tables using the `FEDERATED` storage engine is no longer permitted. Attempting to create such a table or to modify an existing table so that it uses both partitioning and `FEDERATED` now fails with an error. (Bug#22451)
- In MySQL 5.1, `InnoDB` rolls back only the last statement on a transaction timeout. A new option, `--innodb_rollback_on_timeout`, causes `InnoDB` to abort and roll back the entire transaction if a transaction timeout occurs (the same behavior as in MySQL 4.1). (Bug#24200)
- The `Com_create_user` status variable was added (for counting `CREATE USER` statements). (Bug#22958)
- The `--memlock` option relies on system calls that are unreliable on some operating systems. If a crash occurs, the server now checks whether `--memlock` was specified and if so issues some information about possible workarounds. (Bug#22860)
- The default value of the `max_connections` variable has been increased to 151 in order that Websites running on Apache and using MySQL will not have more processes trying to access MySQL than the default number of connections available.

(The maximum number of Apache processes is determined by the Apache `MaxClient`, which defaults to 256, but is usually set to 150 in the `httpd.conf` commonly distributed with Apache. For more information about `MaxClient`, see http://httpd.apache.org/docs/2.2/mod/mpm_common.html#maxclients.)

(Bug#23883)
- The bundled `yaSSL` library was upgraded to version 1.5.0.
- Calling a non-deterministic stored routine when using statement-based replication now throws an error. Formerly, defining such a stored routine would cause an error to be thrown. (Bug#16456)
- The (undocumented) `UNIQUE_USERS()` and `GROUP_UNIQUE_USERS()` functions were removed. (Bug#22687)
- **NDB Cluster:** The `LockPagesInMainMemory` configuration parameter has changed its type and possible values. For more information, see `LockPagesInMainMemory`. (Bug#25686)

Important

The values `true` and `false` are no longer accepted for this parameter. If you were using this parameter and had it set to `false` in a previous release, you must change it to `0`. If you had this parameter set to `true`, you should instead use `1` to obtain the same behavior as previously, or `2` to take advantage of new functionality introduced with this release described in the section cited above.

Bugs fixed:

- **Incompatible change:** For `ENUM` columns that had enumeration values containing commas, the commas were mapped to `0xff` internally. However, this rendered the commas indistinguishable from true `0xff` characters in the values. This no longer occurs. However, the fix requires that you dump and reload any tables that have `ENUM` columns containing true `0xff` in their values: Dump the tables using `mysqldump` with the current server before upgrading from a version of MySQL 5.1 older than 5.1.15 to version 5.1.15 or newer. (Bug#24660)
- `mysqld_multi` and `mysqlaccess` looked for option files in `/etc` even if the `--sysconfdir` option for `configure` had been given to specify a different directory. (Bug#24780)
- `SHOW COLUMNS` reported some `NOT NULL` columns as `NULL`. (Bug#22377)

- Attempts to access a **MyISAM** table with a corrupt column definition caused a server crash. ([Bug#24401](#))
- **mysqltest_embedded** crashed at startup. ([Bug#25890](#))
- Accessing a fixed record format table with a crashed key definition results in server/**myisamchk** segmentation fault. ([Bug#24855](#))
- When opening a corrupted **.frm** file during a query, the server crashes. ([Bug#24358](#))
- If there was insufficient memory to store or update a blob record in a **MyISAM** table then the table will marked as crashed. ([Bug#23196](#))
- When updating a table that used a **JOIN** of the table itself (for example, when building trees) and the table was modified on one side of the expression, the table would either be reported as crashed or the wrong rows in the table would be updated. ([Bug#21310](#))
- When **SET PASSWORD** was written to the binary log double quotes were included in the statement. If the slave was running in with the **sql_mode** set to **ANSI_QUOTES** the event would fail and halt the replication process. ([Bug#24158](#))
- Using **CREATE TABLE ... SELECT** and rolling back the transaction would leave an empty table on the master, but the instructions would not be recorded in the binary log and therefore replicated to the slave. This would result in a difference between the master and slave databases. An implicit commit has been added to ensure consistency. ([Bug#22865](#))
- When reading from the standard input on Windows, **mysqlbinlog** opened the input in text mode rather than binary mode and consequently misinterpreted some characters such as Control-Z. ([Bug#23735](#))
- No warning was issued for use of the **DATA DIRECTORY** or **INDEX DIRECTORY** table options on a platform that does not support them. ([Bug#17498](#))
- When a prepared statement failed during the prepare operation, the error code was not cleared when it was reused, even if the subsequent use was successful. ([Bug#15518](#))
- **mysql_upgrade** failed when called with a **basedir** pathname containing spaces. ([Bug#22801](#))
- Hebrew-to-Unicode conversion failed for some characters. Definitions for the following Hebrew characters (as specified by the ISO/IEC 8859-8:1999) were added: LEFT-TO-RIGHT MARK (LRM), RIGHT-TO-LEFT MARK (RLM) ([Bug#24037](#))
- If there was insufficient memory available to **mysqld**, this could sometimes cause the server to hang during startup. ([Bug#24751](#))
- Using row-based replication to replicate to a table having at least one extra **BIT** column with a default value on the slave as compared to the master could cause the slave to fail. ([Bug#24490](#))
- Optimizations that are legal only for subqueries without tables and **WHERE** conditions were applied for any subquery without tables. ([Bug#24670](#))
- A query using **WHERE unsigned_column NOT IN ('negative_value')** could cause the server to crash. ([Bug#24261](#))
- A **FETCH** statement using a cursor on a table which was not in the table cache could sometimes cause the server to crash. ([Bug#24117](#))
- **CREATE TABLE ... SELECT** statements were not rolled back correctly. As part of the fix, such a statement now causes an implicit commit before and after it is executed. However, it does not cause a commit when used to create a temporary table. ([Bug#22864](#))
- SSL connections could hang at connection shutdown. ([Bug#24148](#), [Bug#21781](#))
- The **STDDEV()** function returned a positive value for data sets consisting of a single value. ([Bug#22555](#))
- **mysqltest** incorrectly tried to retrieve result sets for some queries where no result set was available. ([Bug#19410](#))
- **mysqltest** crashed with a stack overflow. ([Bug#24498](#))
- The server was built even when **configure** was run with the **--without-server** option. ([Bug#23973](#))
- **mysqld_error.h** was not installed when only the client libraries were built. ([Bug#21265](#))
- **mysql_install_db** did not create the **mysql.plugin** table if strict SQL mode was enabled. ([Bug#24270](#))

- The row count for **MyISAM** tables was not updated properly, causing **SHOW TABLE STATUS** to report incorrect values. ([Bug#23526](#))
- Using a view in combination with a **USING** clause caused column aliases to be ignored. ([Bug#25106](#))
- Using **ALTER TABLE** to convert a **CSV** table containing **NULL** values to **MyISAM** resulted in warnings. ([Bug#21328](#))
- A view was not handled correctly if the **SELECT** part contained `'\Z'`. ([Bug#24293](#))
- Inserting a row into a table without specifying a value for a **BINARY(N) NOT NULL** column caused the column to be set to spaces, not zeroes. ([Bug#14171](#))
- An assertion failed incorrectly for prepared statements that contained a single-row uncorrelated subquery that was used as an argument of the **IS NULL** predicate. ([Bug#25027](#))
- A table created with the **ROW_FORMAT = FIXED** table option loses the option if an index is added or dropped with **CREATE INDEX** or **DROP INDEX**. ([Bug#23404](#))
- Dropping a user-defined function sometimes did not remove the UDF entry from the `mysql.proc` table. ([Bug#15439](#))
- The **BUILD/check-cpu** script did not recognize Celeron processors. ([Bug#20061](#))
- Some problems uncovered by Valgrind were fixed. ([Bug#25396](#))
- `mysql_fix_privilege_tables` did not handle a password containing embedded space or apostrophe characters. ([Bug#17700](#))
- On Windows, the **SLEEP()** function could sleep too long, especially after a change to the system clock. ([Bug#14094](#), [Bug#17635](#), [Bug#24686](#))
- In the **INFORMATION_SCHEMA.KEY_COLUMN_USAGE** table, the value displayed for the **REFERENCED_TABLE_NAME** column was the table name as encoded for disk storage, not the actual table name. ([Bug#25026](#))
- Changing the value of **MI_KEY_BLOCK_LENGTH** in `myisam.h` and recompiling MySQL resulted in a `myisamchk` that saw existing **MyISAM** tables as corrupt. ([Bug#22119](#))
- A stored routine containing semicolon in its body could not be reloaded from a dump of a binary log. ([Bug#20396](#))
- For **SET**, **SELECT**, and **DO** statements that invoked a stored function from a database other than the default database, the function invocation could fail to be replicated. ([Bug#19725](#))
- **SET lc_time_names = value** allowed only exact literal values, not expression values. ([Bug#22647](#))
- Changes to the `lc_time_names` system variable were not replicated. ([Bug#22645](#))
- **SELECT ... FOR UPDATE**, **SELECT ... LOCK IN SHARE MODE**, **DELETE**, and **UPDATE** statements executed using a full table scan were not releasing locks on rows that did not satisfy the **WHERE** condition. ([Bug#20390](#))
- A stored procedure, executed from a connection using a binary character set, and which wrote multibyte data, would write incorrectly escaped entries to the binary log. This caused syntax errors, and caused replication to fail. ([Bug#23619](#), [Bug#24492](#))
- `mysqldump --order-by-primary` failed if the primary key name was an identifier that required quoting. ([Bug#13926](#))
- Re-execution of **CREATE DATABASE**, **CREATE TABLE**, and **ALTER TABLE** statements in stored routines or as prepared statements caused incorrect results. ([Bug#22060](#))
- A workaround was implemented to avoid a race condition in the NPTL `pthread_exit()` implementation. ([Bug#24507](#))
- The Instance Manager **DROP INSTANCE** command did not work. ([Bug#23476](#))
- The Instance Manager **STOP INSTANCE** command took too much time and caused Instance Manager to be unresponsive. ([Bug#23215](#))
- The Instance Manager **STOP INSTANCE** command could not be applied to instances in the **Crashed**, **Failed**, or **Abandoned** state. ([Bug#22306](#))

- Instance Manager could crash during shutdown. ([Bug#19044](#))
- A deadlock could occur, with the server hanging on `Closing tables`, with a sufficient number of concurrent `INSERT DELAYED`, `FLUSH TABLES`, and `ALTER TABLE` operations. ([Bug#23312](#))
- A user-defined variable could be assigned an incorrect value if a temporary table was employed in obtaining the result of the query used to determine its value. ([Bug#16861](#))
- The optimizer removes expressions from `GROUP BY` and `DISTINCT` clauses if they happen to participate in `expression = constant` predicates of the `WHERE` clause, the idea being that, if the expression is equal to a constant, then it cannot take on multiple values. However, for predicates where the expression and the constant item are of different result types (for example, when a string column is compared to 0), this is not valid, and can lead to invalid results in such cases. The optimizer now performs an additional check of the result types of the expression and the constant; if their types differ, then the expression is not removed from the `GROUP BY` list. ([Bug#15881](#))
- Referencing an ambiguous column alias in an expression in the `ORDER BY` clause of a query caused the server to crash. ([Bug#25427](#))
- Some `CASE` statements inside stored routines could lead to excessive resource usage or a crash of the server. ([Bug#24854](#), [Bug#19194](#))
- Some joins in which one of the joined tables was a view could return erroneous results or crash the server. ([Bug#24345](#))
- `OPTIMIZE TABLE` tried to sort R-tree indexes such as spatial indexes, although this is not possible (see [Section 12.5.2.5, “OPTIMIZE TABLE Syntax”](#)). ([Bug#23578](#))
- User-defined variables could consume excess memory, leading to a crash caused by the exhaustion of resources available to the `MEMORY` storage engine, due to the fact that this engine is used by MySQL for variable storage and intermediate results of `GROUP BY` queries. Where `SET` had been used, such a condition could instead give rise to the misleading error message `YOU MAY ONLY USE CONSTANT EXPRESSIONS WITH SET`, rather than `OUT OF MEMORY (NEEDED NNNNNN BYTES)`. ([Bug#23443](#))
- `InnoDB`: During a restart of the MySQL Server that followed the creation of a temporary table using the `InnoDB` storage engine, MySQL failed to clean up in such a way that `InnoDB` still attempted to find the files associated with such tables. ([Bug#20867](#))
- Under some circumstances, a `REORGANIZE PARTITION` statement could crash `mysqld`. ([Bug#24502](#))
- A multiple-table `DELETE QUICK` could sometimes cause one of the affected tables to become corrupted. ([Bug#25048](#))
- A compressed `MyISAM` table that became corrupted could crash `myisamchk` and possibly the MySQL Server. ([Bug#23139](#))
- Using `INSTALL PLUGIN` followed by a restart of the server caused an error due to memory not being properly initialized. ([Bug#22694](#))
- A crash of the MySQL Server could occur when unpacking a `BLOB` column from a row in a corrupted `MyISAM` table. This could happen when trying to repair a table using either `REPAIR TABLE` or `myisamchk`; it could also happen when trying to access such a “broken” row using statements like `SELECT` if the table was not marked as crashed. ([Bug#22053](#))
- The `FEDERATED` storage engine did not support the `euckr` character set. ([Bug#21556](#))
- The `FEDERATED` storage engine did not support the `utf8` character set. ([Bug#17044](#))
- `mysql_upgrade` failed if the `--password` (or `-p`) option was given. ([Bug#24896](#))
- For a nonexistent table, `DROP TEMPORARY TABLE` failed with an incorrect error message if `read_only` was enabled. ([Bug#22077](#))
- The code for generating `USE` statements for binary logging of `CREATE PROCEDURE` statements resulted in confusing output from `mysqlbinlog` for `DROP PROCEDURE` statements. ([Bug#22043](#))
- The `REPEAT ( )` function could return `NULL` when passed a column for the count argument. ([Bug#24947](#))
- Accuracy was improved for comparisons between `DECIMAL` columns and numbers represented as strings. ([Bug#23260](#))
- `InnoDB` crashed while performing XA recovery of prepared transactions. ([Bug#21468](#))
- `ROW_COUNT ( )` did not work properly as an argument to a stored procedure. ([Bug#23760](#))

- It was possible to use `DATETIME` values whose year, month, and day parts were all zeroes but whose hour, minute, and second parts contained nonzero values, an example of such an illegal `DATETIME` being `'0000-00-00 11:23:45'`. (Bug#21789)

This patch was reverted in MySQL 5.1.18.

- It was possible to set the backslash character (`" \ "`) as the delimiter character using `DELIMITER`, but not actually possible to use it as the delimiter. (Bug#21412)
- `ALTER ENABLE KEYS` or `ALTER TABLE DISABLE KEYS` combined with another `ALTER TABLE` option other than `RENAME TO` did nothing. In addition, if `ALTER TABLE` was used on a table having disabled keys, the keys of the resulting table were enabled. (Bug#24395)
- An `ALTER TABLE` statement that used a `RENAME` clause in combination with a `MODIFY` or `CHANGE` that did not actually change the table (for example, when it changed a column's type from `INT` to `INT`). The behavior caused by this bug differed according to whether or not the storage engine used by the table was transactional or non-transactional. For transactional tables (such as those using the `InnoDB` storage engine), the statement simply failed; for non-transactional tables (such as those using the `MyISAM` storage engine), the `ALTER TABLE` statement succeeding renaming the table, but subsequent `SELECT` statements against the renamed table would fail. (Bug#22369)
- Queries of the form `SELECT ... WHERE string = ANY(...)` failed when the server used a single-byte character set and the client used a multi-byte character set. (Bug#20835)
- A partitioned table that used the `DATA DIRECTORY` option, where the data directory was the same as the directory in which the table definition file resided, became corrupted following `ALTER TABLE ENGINE=ARCHIVE`. This was actually due to an issue with the `ARCHIVE` storage engine, and not with partitioned tables in general. (Bug#22634)
- `NDB Cluster` (Cluster APIs): Deletion of an `Ndb_cluster_connection` object took a very long time. (Bug#25487)
- `NDB Cluster` (Replication): Certain errors in replication setups could lead to subsequent node failures. (Bug#25755)
- `NDB Cluster` (Replication): Connecting a `mysqld` to a cluster where not all nodes were running, starting the remaining cluster nodes, and then disconnecting from the cluster caused the `mysqld` process to crash. (Bug#25387)
- `NDB Cluster` (Replication): Connecting an API node to the cluster during a node restart while performing database operations could cause the restarting node to fail. (Bug#25329)
- `NDB Cluster` (Disk Data): Following 3 or more missed local checkpoints by a cluster node, a restart of the node caused incorrect undo information to be used for Disk Data tables. (Bug#25636)
- `NDB Cluster`: It was not possible to create an `NDB` table with a key on two `VARCHAR` columns where both columns had a storage length in excess of 256. (Bug#25746)
- `NDB Cluster`: Hosts in clusters with a large number of nodes could experience excessive CPU usage while obtaining configuration data. (Bug#25711)
- `NDB Cluster`: In some circumstances, shutting down the cluster could cause connected `mysqld` processes to crash. (Bug#25668)
- `NDB Cluster`: Non-32-bit, non-aligned columns were not handled correctly in explicitly partitioned `NDB` tables. (Bug#25587)
- `NDB Cluster`: Some aggregate queries such as `SELECT COUNT(*)` performed a table scan on `NDB` tables rather than checking table statistics, causing such queries to perform much more slowly in MySQL Cluster 5.1 than in 5.0. (Bug#25567)
- `NDB Cluster`: Memory allocations for `TEXT` columns were calculated incorrectly, resulting in space being wasted and other issues. (Bug#25562)
- `NDB Cluster`: The failure of a master node during a node restart could lead to a resource leak, causing later node failures. (Bug#25554)
- `NDB Cluster`: The failure of a node during a local checkpoint could lead to other node failures. (Bug#25468)
- `NDB Cluster`: A node shutdown occurred if the master failed during a commit. (Bug#25364)
- `NDB Cluster`: Creating a non-unique index with the `USING HASH` clause silently created an ordered index instead of issuing a warning. (Bug#24820)

- **NDB Cluster**: The management server did not handle logging of node shutdown events correctly in certain cases. ([Bug#22013](#))
- **NDB Cluster**: A potential memory leak in the **NDB** storage engine's handling of file operations was uncovered. ([Bug#21858](#))
- **NDB Cluster**: When stopping and restarting multiple data nodes, the last node to be restarted would sometimes hang in Phase 100. ([Bug#19645](#))
- **NDB Cluster** (NDB API): Invoking the `NdbTransaction::execute()` method using execution type **Commit** and abort option **AO_IgnoreError** could lead to a crash of the transaction coordinator (DBTC). ([Bug#25090](#))
- **NDB Cluster** (NDB API): A unique index lookup on a non-existent tuple could lead to a data node timeout (error 4012). ([Bug#25059](#))
- **NDB Cluster** (NDB API): Due to an error in the computation of table fragment arrays, some transactions might not be executed from the correct starting point. ([Bug#24914](#))
- **NDB Cluster** (Replication): Following a restart of the master cluster, the latest GCI was set to 0 upon reconnection to the slave. ([Bug#21806](#))
- **NDB Cluster** (Disk Data): A **MEDIUMTEXT** column of a Disk Data table was stored in memory rather than on disk, even if the column was not indexed. ([Bug#25001](#))
- **NDB Cluster** (Disk Data): Performing a node restart with a newly dropped Disk Data table could lead to failure of the node during the restart. ([Bug#24917](#))
- **NDB Cluster** (Disk Data): When restoring from backup a cluster containing any Disk Data tables with hidden primary keys, a node failure resulted which could lead to a crash of the cluster. ([Bug#24166](#))
- **NDB Cluster** (Disk Data): Repeated **CREATE**, **DROP**, or **TRUNCATE** in various combinations with system restarts between these operations could lead to the eventual failure of a system restart. ([Bug#21948](#))
- **NDB Cluster** (Disk Data): Extents that should have been available for re-use following a **DROP TABLE** operation were not actually made available again until after the cluster performed a local checkpoint. ([Bug#17605](#))
- **NDB Cluster**: Under certain rare circumstances, local checkpoints were not performed properly, leading to an inability to restart one or more data nodes. ([Bug#24664](#))
- **NDB Cluster** (NDB API): When using the `NdbTransaction::execute()` method, a very long timeout (greater than 5 minutes) could result if the last data node being polled was disconnected from the cluster. ([Bug#24949](#))
- **NDB Cluster**: When a data node was shut down using the management client **STOP** command, a connection event (**NDB_LE_Connected**) was logged instead of a disconnection event (**NDB_LE_Disconnected**). ([Bug#22773](#))
- **NDB Cluster**: **SELECT** statements with a **BLOB** or **TEXT** column in the selected column list and a **WHERE** condition including a primary key lookup on a **VARCHAR** primary key produced empty result sets. ([Bug#19956](#))
- **NDB Cluster**: `ndb_config` failed when trying to use 2 management servers and node IDs. ([Bug#23887](#))
- **STR_TO_DATE()** returned **NULL** if the format string contained a space following a non-format character. ([Bug#22029](#))
- **yaSSL** crashed on pre-Pentium Intel CPUs. ([Bug#21765](#))
- Selecting into variables sometimes returned incorrect wrong results. ([Bug#20836](#))
- `mysql_fix_privilege_tables.sql` altered the `table_privs.table_priv` column to contain too few privileges, causing loss of the **CREATE VIEW** and **SHOW VIEW** privileges. ([Bug#20589](#))
- A server crash occurred when using **LOAD DATA** to load a table containing a **NOT NULL** spatial column, when the statement did not load the spatial column. Now a **NULL supplied to NOT NULL column** error occurs. ([Bug#22372](#))
- Unsigned **BIGINT** values treated as signed values by the **MOD()** function. ([Bug#19955](#))
- Compiling PHP 5.1 with the MySQL static libraries failed on some versions of Linux. ([Bug#19817](#))
- The **DELIMITER** statement did not work correctly when used in an SQL file run using the **SOURCE** statement. ([Bug#19799](#))

- `VARBINARY` column values inserted on a MySQL 4.1 server had trailing zeroes following upgrade to MySQL 5.0 or later. (Bug#19371)
- Subqueries of the form `NULL IN (SELECT ...)` returned invalid results. (Bug#8804, Bug#23485)
- The `--extern` option for `mysql-test-run.pl` did not function correctly. (Bug#24354)
- `ALTER TABLE` statements that performed both `RENAME TO` and `{ENABLE|DISABLE} KEYS` operations caused a server crash. (Bug#24089)
- The MySQL 5.1.12 binaries for Windows were missing the `FEDERATED`, `EXAMPLE`, and `BLACKHOLE` storage engines. (Bug#23900)
- `myisampack` wrote to unallocated memory, causing a crash. (Bug#17951)
- Some small double precision numbers (such as `1.00000001e-300`) that should have been accepted were truncated to zero. (Bug#22129)
- The `mysql.server` script used the `source` command, which is less portable than the `.` command; it now uses `.` instead. (Bug#24294)
- `DATE_ADD()` requires complete dates with no “zero” parts, but sometimes did not return `NULL` when given such a date. (Bug#22229)
- `FLUSH LOGS` or `mysqladmin flush-logs` caused a server crash if the binary log was not open. (Bug#17733)
- On HP-UX, `mysqltest` (non-thread-safe) crashed due to being linked against a thread-safe `libmysys` library. (Bug#23984)

C.1.14. Changes in MySQL 5.1.15 Carrier Grade Edition

This section contains change history information for MySQL Cluster 5.1 Carrier Grade Edition releases based on MySQL 5.1.15.

C.1.14.1. Changes in release MySQL 5.1.15-ndb-6.1.19-beta (01 August 2007)

This is a new Beta development release, incorporating recent feature enhancements since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.19/. The file `mysqlcom-5.1.15-ndb-6.1.19-telco.tar.gz` in this directory contains the complete source archive.

Note

Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, MySQL 5.1.15-ndb-6.1.8, MySQL 5.1.15-ndb-6.1.9, MySQL 5.1.15-ndb-6.1.10, MySQL 5.1.15-ndb-6.1.11, MySQL 5.1.15-ndb-6.1.12, MySQL 5.1.15-ndb-6.1.13, MySQL 5.1.15-ndb-6.1.14, MySQL 5.1.15-ndb-6.1.15, MySQL 5.1.15-ndb-6.1.16, MySQL 5.1.15-ndb-6.1.17, and MySQL 5.1.15-ndb-6.1.18. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

Functionality added or changed:

- An **INFO** event is now sent if the time between global checkpoints is excessive, or if **DUMP 7901** is issued in the management client.
- Whenever a TCP send buffer is over 80% full, temporary error 1218 (**SEND BUFFERS OVERLOADED IN NDB KERNEL**) is now re-

turned. See [SendBufferMemory](#) for more information.

C.1.14.2. Changes in release MySQL 5.1.15-ndb-6.1.18-beta (Not released)

This is a new Beta development release, fixing recently discovered bugs in the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Note

Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, MySQL 5.1.15-ndb-6.1.8, MySQL 5.1.15-ndb-6.1.9, MySQL 5.1.15-ndb-6.1.10, MySQL 5.1.15-ndb-6.1.11, MySQL 5.1.15-ndb-6.1.12, MySQL 5.1.15-ndb-6.1.13, MySQL 5.1.15-ndb-6.1.14, MySQL 5.1.15-ndb-6.1.15, MySQL 5.1.15-ndb-6.1.16, and MySQL 5.1.15-ndb-6.1.17. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

This release fixes the following bugs:

- **Replication:** Storage engine error conditions in row-based replication were not correctly reported to the user. ([Bug#29570](#))
- A problem with the fix for [Bug#29354](#) caused an assertion when two local checkpoints were run during node recovery.
- **Disk Data:** Disk data meta-information that existed in `ndbnd` might not be visible to `mysqld`. ([Bug#28720](#))
- **Disk Data:** The number of free extents was incorrectly reported for some tablespaces. ([Bug#28642](#))
- When restarting a data node, queries could hang during that node's start phase 5, and only continue once the node entered phase 6. ([Bug#29364](#))

C.1.14.3. Changes in release MySQL 5.1.15-ndb-6.1.17-beta (Released 03 July 2007)

This is a new Beta development release, fixing recently discovered bugs in the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.17/. The file `mysqlcom-5.1.15-ndb-6.1.17-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, MySQL 5.1.15-ndb-6.1.8, MySQL 5.1.15-ndb-6.1.9, MySQL 5.1.15-ndb-6.1.10, MySQL 5.1.15-ndb-6.1.11, MySQL 5.1.15-ndb-6.1.12, MySQL 5.1.15-ndb-6.1.13, MySQL 5.1.15-ndb-6.1.14, MySQL 5.1.15-ndb-6.1.15, and MySQL 5.1.15-ndb-6.1.16. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

Functionality added or changed:

- Batching of updates on cluster replication slaves, enabled using the `--slave-allow-batching` option for `mysqld`.

This release fixes the following bug:

- Replica redo logs were inconsistently handled during a system restart. ([Bug#29354](#))

C.1.14.4. Changes in release MySQL 5.1.15-ndb-6.1.16-beta (Released 29 June 2007)

This is a new Beta development release, fixing recently discovered bugs in the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.16/. The file `mysqlcom-5.1.15-ndb-6.1.16-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, MySQL 5.1.15-ndb-6.1.8, MySQL 5.1.15-ndb-6.1.9, MySQL 5.1.15-ndb-6.1.10, MySQL 5.1.15-ndb-6.1.11, MySQL 5.1.15-ndb-6.1.12, MySQL 5.1.15-ndb-6.1.13, MySQL 5.1.15-ndb-6.1.14, and MySQL 5.1.15-ndb-6.1.15. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

This release fixes the following bugs:

- If at least 2 files were involved in REDO invalidation, then file 0 of page 0 was not updated and so pointed to an invalid part of the redo log. ([Bug#29057](#))
- The wrong data pages were sometimes invalidated following a global checkpoint. ([Bug#29067](#))
- An invalid comparison made during REDO validation that could lead to an `ERROR WHILE READING REDO LOG` condition. ([Bug#29118](#))
- **Disk Data:** When dropping a page, the stack's bottom entry could sometime be left “cold” rather than “hot”, violating the rules for stack pruning. ([Bug#29176](#))

Note

This supersedes an earlier fix for this bug that was made in MySQL 5.1.15-ndb-6.1.14.

- When a node failed to respond to a `COPY_GCI` signal as part of a global checkpoint, the master node was killed instead instead of the node that actually failed. ([Bug#29331](#))

C.1.14.5. Changes in release MySQL 5.1.15-ndb-6.1.15-beta (Released 20 June 2007)

This is a new Beta development release, fixing a recently discovered bug in the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.15/. The file `mysqlcom-5.1.15-ndb-6.1.15-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, MySQL 5.1.15-ndb-6.1.8, MySQL 5.1.15-ndb-6.1.9, MySQL 5.1.15-ndb-6.1.10, MySQL 5.1.15-ndb-6.1.11, MySQL

5.1.15-ndb-6.1.12, MySQL 5.1.15-ndb-6.1.13, and MySQL 5.1.15-ndb-6.1.14. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

This release fixes the following bug:

- Memory corruption could occur due to a problem in the [DBTUP](#) kernel block. ([Bug#29229](#))

C.1.14.6. Changes in release MySQL 5.1.15-ndb-6.1.14-beta (Released 19 June 2007)

This is a new Beta development release, incorporating bugfixes made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.14/. The file `mysqlcom-5.1.15-ndb-6.1.14-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, MySQL 5.1.15-ndb-6.1.8, MySQL 5.1.15-ndb-6.1.9, MySQL 5.1.15-ndb-6.1.10, MySQL 5.1.15-ndb-6.1.11, MySQL 5.1.15-ndb-6.1.12, and MySQL 5.1.15-ndb-6.1.13. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

This release fixes the following bugs:

- In the event that two data nodes in the same node group and participating in a GCP crashed before they had written their respective `P0.sysfile` files, `QMGR` could refuse to start, issuing an invalid `INSUFFICIENT NODES FOR RESTART` error instead. ([Bug#29167](#))
- **Disk Data:** When dropping a page, the stack's bottom entry could sometime be left “cold” rather than “hot”, violating the rules for stack pruning. ([Bug#29176](#))

C.1.14.7. Changes in release MySQL 5.1.15-ndb-6.1.13-beta (Released 15 June 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.13/. The file `mysqlcom-5.1.15-ndb-6.1.13-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, MySQL 5.1.15-ndb-6.1.8, MySQL 5.1.15-ndb-6.1.9, MySQL 5.1.15-ndb-6.1.10, MySQL 5.1.15-ndb-6.1.11, and MySQL 5.1.15-ndb-6.1.12. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

Functionality added or changed:

- Read ahead was implemented for backups of Disk Data tables, resulting in a 10 to 15% increase in backup speed of Disk Data tables. ([Bug#29099](#))

C.1.14.8. Changes in release MySQL 5.1.15-ndb-6.1.12-beta (Released 13 June 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.12/. The file `mysqlcom-5.1.15-ndb-6.1.12-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, MySQL 5.1.15-ndb-6.1.8, MySQL 5.1.15-ndb-6.1.9, MySQL 5.1.15-ndb-6.1.10, and MySQL 5.1.15-ndb-6.1.11. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

Functionality added or changed:

- It is now possible to set the maximum size of the allocation unit for table memory using the `MaxAllocate` configuration parameter. ([Bug#29044](#))
- Backup dump output was extended to provide more information.
- New cluster management client `DUMP` commands were added to aid in tracking transactions, scan operations, and locks. See `DUMP 2350` and `DUMP 2550`.

C.1.14.9. Changes in release MySQL 5.1.15-ndb-6.1.11-beta (Released 06 June 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.11/. The file `mysqlcom-5.1.15-ndb-6.1.11-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, MySQL 5.1.15-ndb-6.1.8, MySQL 5.1.15-ndb-6.1.9, and MySQL 5.1.15-ndb-6.1.10. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

Functionality added or changed:

- It is now possible to set the size of redo log files (fragment log files) using the `FragmentLogFileSize` configuration parameter.
- A new configuration parameter `ODirect` causes `NDB` to attempt using `O_DIRECT` writes for LCP, backups, and redo logs, often lowering CPU usage.

This release fixes the following bugs:

- Having large amounts of memory locked caused swapping to disk. ([Bug#28751](#))
- It was not possible to set a separate watchdog timeout at startup. ([Bug#28899](#))
- LCP files were not removed following an initial system restart. ([Bug#28726](#))
- **Disk Data:** Repeated **INSERT** and **DELETE** operations on a Disk Data table having one or more large **VARCHAR** columns could cause data nodes to fail. ([Bug#20612](#))

C.1.14.10. Changes in release MySQL 5.1.15-ndb-6.1.10-beta (Released 30 May 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.10/. The file `mysqlcom-5.1.15-ndb-6.1.10-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, MySQL 5.1.15-ndb-6.1.8, and MySQL 5.1.15-ndb-6.1.9. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

Functionality added or changed:

- A new `times` printout was added in the `ndbd` watchdog thread.
- The names of some log and other files were changed to avoid issues with the `tar` command's 99-character filename limit.
- Some unneeded printouts in the `ndbd` out file were removed.

This release fixes the following bugs:

- Heartbeat failures could occur under high load. ([Bug#28783](#))
- A corrupt schema file could cause a `FILE ALREADY OPEN` error. ([Bug#28770](#))
- A race condition could occur between `NODE_FAILREP` and `COPY_GCIREQ` events. ([Bug#28717](#))
- While loading data to the cluster after a version upgrade, the data nodes failed due to `ndbrequire` failures in `PGMAN`. ([Bug#28525](#))
- A fast global checkpoint under high load with a high usage of the redo buffer caused data nodes to fail. ([Bug#28653](#))
- The actual value of `MaxNoOfOpenFiles` as used by the cluster was offset by 1 from the value set in `config.ini`. This meant that setting `InitialNoOpenFiles` to the same value always caused an error. ([Bug#28749](#))

C.1.14.11. Changes in release MySQL 5.1.15-ndb-6.1.9-beta (Released 24 May 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.9/. The file `mysqlcom-5.1.15-ndb-6.1.9-telco.tar.gz` in this directory contains the complete source archive.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, MySQL 5.1.15-ndb-6.1.7, and MySQL 5.1.15-ndb-6.1.8. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

This release fixes the following bugs:

- The cluster backup process scanned in [ACC](#) index order, which had bad effects for disk data. ([Bug#28593](#))
- The [NDB](#) transporter would hang when more than 1024 signals were received at one time. ([Bug#28443](#))

C.1.14.12. Changes in release MySQL 5.1.15-ndb-6.1.8-beta (Released 05 May 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.8/. The file `mysqlcom-5.1.15-ndb-6.1.8-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, MySQL 5.1.15-ndb-6.1.6, and MySQL 5.1.15-ndb-6.1.7. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

This release fixes the following bugs:

- Setting [MaxNoOfTables](#) very low and relative to [DataMemory](#) caused `OUT OF MEMORY IN NDB KERNEL` errors when inserting relatively small amounts of data into NDB tables. ([Bug#24173](#))
- Local checkpoint files related to dropped [NDB](#) tables were not removed. ([Bug#28348](#))
- **Disk Data:** Extremely large inserts into Disk Data tables could lead to data node failure in some circumstances. ([Bug#27942](#))
- Repeated insertion of data generated by `mysqldump` into [NDB](#) tables could eventually lead to failure of the cluster. ([Bug#27437](#))
- **Cluster APIs:** In a multi-operation transaction, a delete operation followed by the insertion of an implicit [NULL](#) failed to overwrite an existing value. ([Bug#20535](#))

C.1.14.13. Changes in release MySQL 5.1.15-ndb-6.1.7-beta (Released 05 May 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL

5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, MySQL 5.1.15-ndb-6.1.5, and MySQL 5.1.15-ndb-6.1.6. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13](#), “Changes in release 5.1.15 (25 January 2007)”).

Important

Upgrading to MySQL 5.1.15-ndb-6.1.7 from a previous release. The internal specifications for columns in **NDB** tables has changed to allow compatibility with future MySQL Cluster and MySQL Cluster 5.1 Carrier Grade Edition releases that are expected to implement online adding and dropping of columns. This change is not backwards compatible with MySQL 5.1.16-ndb-6.2.0, ndb-6.1.x versions prior to MySQL 5.1.15-ndb-6.1.7, or MySQL Cluster mainline releases prior to 5.1.18.

See [the related note](#) in [Section 16.5.2](#), “Cluster Upgrade and Downgrade Compatibility”, for important information prior to upgrading a MySQL Cluster to MySQL 5.1.15-ndb-6.1.7 or later from MySQL 5.1.15-ndb-6.1.6 or an earlier ndb-6.1.x release.

See also [Bug#28205](#).

Functionality added or changed:

- **Incompatible change:** The schema for the `ndb_apply_status` table in the `mysql` system database has changed. When upgrading to this release from a previous MySQL Cluster 5.1 Carrier Grade Edition or mainline MySQL 5.1 release, you must drop the `mysql.ndb_apply_status` table, then restart the server in order for the table to be re-created with the new schema.

See [Section 16.10.4](#), “Cluster Replication Schema and Tables”, for additional information.

This release fixes the following bugs:

- Under certain rare circumstances, `DROP TABLE` or `TRUNCATE` of an **NDB** table could cause a node failure or forced cluster shutdown. ([Bug#27581](#))
- Memory usage of a `mysqld` process grew even while idle. ([Bug#27560](#))
- **Cluster APIs:** An issue with the way in which the `NdbDictionary::Dictionary::listEvents()` method freed resources could sometimes lead to memory corruption. ([Bug#27663](#))
- **Cluster Replication:** Under very high loads, checkpoints could be read or written with checkpoint indexes out of order. ([Bug#27651](#))
- **Cluster Replication:** It was possible for API nodes to begin interacting with the cluster subscription manager before they were fully connected to the cluster. ([Bug#27728](#))
- Some queries that updated multiple tables were not backed up or replicated correctly. ([Bug#27748](#))
- **Disk Data:** Changes to a Disk Data table made as part of a transaction could not be seen by the client performing the changes until the transaction had been committed. ([Bug#27757](#))
- An `INSERT` followed a delete `DELETE` on the same **NDB** table caused a memory leak. ([Bug#27756](#))
(This bug was introduced by the fix for [Bug#20612](#).)
- Under some circumstances, a node restart could fail to update the Global Checkpoint Index (GCI). ([Bug#28023](#))
- Under certain rare circumstances, `ndbd` could get caught in an infinite loop when one transaction took a read lock and then a second transaction attempted to obtain a write lock on the same tuple in the lock queue. ([Bug#28073](#))
- The cluster waited 30 seconds instead of 30 milliseconds before reading table statistics. ([Bug#28093](#))
- **Disk Data:** Changing a column specification or issuing a `TRUNCATE` statement on a Disk Data table caused the table to become an in-memory table. ([Bug#24667](#))
- Performing a delete followed by an insert during a local checkpoint could cause a `ROWID ALREADY ALLOCATED` error. ([Bug#27205](#))

- **Disk Data:** When restarting a data node following the creation of a large number (~200) of Disk Data objects, the cluster could not assign a node ID to the restarting node. ([Bug#25741](#))
- **Cluster Replication / Disk Data:** An issue with replication of Disk Data tables could in some cases lead to node failure. ([Bug#28161](#))
- `mysqldump` could not dump log tables. ([Bug#26121](#))
- The `--with-readline` option for `configure` did not work for commercial source packages, but no error message was printed to that effect. Now a message is printed. ([Bug#25530](#))

C.1.14.14. Changes in release MySQL 5.1.15-ndb-6.1.6-beta (Not released)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, MySQL 5.1.15-ndb-6.1.4, and MySQL 5.1.15-ndb-6.1.5. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13](#), “Changes in release 5.1.15 (25 January 2007)”).

Functionality added or changed:

- **Incompatible change:** The schema for the `ndb_apply_status` table in the `mysql` system database has changed. When upgrading to this release from a previous MySQL Cluster 5.1 Carrier Grade Edition or mainline MySQL 5.1 release, you must drop the `mysql.ndb_apply_status` table, then restart the server in order for the table to be re-created with the new schema.

See [Section 16.10.4, “Cluster Replication Schema and Tables”](#), for additional information.

This release fixes the following bugs:

- **Cluster APIs:** An issue with the way in which the `NdbDictionary::Dictionary::listEvents()` method freed resources could sometimes lead to memory corruption. ([Bug#27663](#))
- **Cluster Replication:** Trying to replicate a large number of frequent updates with a relatively small relay log (`max-re-lay-log-size` set to 1M or less) could cause the slave to crash. ([Bug#27529](#))
- A data node failing while another data node was restarting could leave the cluster in an inconsistent state. In certain rare cases, this could lead to a race condition and the eventual forced shutdown of the cluster. ([Bug#27466](#))
- A data node failing while another data node was restarting could leave the cluster in an inconsistent state. In certain rare cases, this could lead to a race condition and the eventual forced shutdown of the cluster. ([Bug#27466](#))
- `mysqld` could crash shortly after a data node failure following certain DML operations. ([Bug#27169](#))
- **Disk Data:** `DROP INDEX` on a Disk Data table did not always move data from memory into the tablespace. ([Bug#25877](#))
- It was not possible to set `LockPagesInMainMemory` equal to 0. ([Bug#27291](#))
- A race condition could sometimes occur if the node acting as master failed while node IDs were still being allocated during startup. ([Bug#27286](#))
- When a data node was taking over as the master node, a race condition could sometimes occur as the node was assuming responsibility for handling of global checkpoints. ([Bug#27283](#))
- A delete operation using a scan following by an insert using a scan could cause a data node to fail. ([Bug#27203](#))
- The same failed request from an API node could be handled by the cluster multiple times, resulting in reduced performance.

[Bug#27087](#))

- `mysqld` processes would sometimes crash under high load. ([Bug#26825](#))

This improves on and replaces the fix for this bug that was made in MySQL 5.1.15-ndb-6.1.5.

- The failure of a data node while restarting could cause other data nodes to hang or crash. ([Bug#27003](#))

C.1.14.15. Changes in release MySQL 5.1.15-ndb-6.1.5-beta (15 March 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.5/. The file `mysqlcom-5.1.15-ndb-6.1.5-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, MySQL 5.1.15-ndb-6.1.3, and MySQL 5.1.15-ndb-6.1.4. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#)).

Important

Upgrading to MySQL 5.1.15-ndb-6.1.5 from a previous release. This release is not binary compatible with releases previous to MySQL 5.1.15-ndb-6.1.3. This means that:

- You cannot perform an online upgrade to this release from MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, or any of the mainline MySQL 5.1 releases. When upgrading from one of these versions, you must shut down the cluster, replace all binaries, then restart the cluster.
- You must recompile all NDB API and MGM API applications used with a previous version of MySQL Cluster, including those compiled against MySQL 5.1.15-ndb-6.1.3.

This release is also not backwards compatible with MySQL 5.1.15-ndb-6.1.14 or any release previous to MySQL 5.1.15-ndb-6.1.14 due to changes in the `mysql` system database. See *Functionality added or changed* for more information.

Functionality added or changed:

- **Incompatible change:** The schema for the `ndb_apply_status` table in the `mysql` system database has changed. When upgrading to this release from a previous MySQL Cluster 5.1 Carrier Grade Edition or mainline MySQL 5.1 release, you must drop the `mysql.ndb_apply_status` table, then restart the server in order for the table to be re-created with the new schema.

See [Section 16.10.4, “Cluster Replication Schema and Tables”](#), for additional information.

This release fixes the following bugs:

- An infinite loop in an internal logging function could cause trace logs to fill up with `UNKNOWN SIGNAL TYPE` error messages and thus grow to unreasonable sizes. ([Bug#26720](#))
- Creating a table on one SQL node while in single user mode caused other SQL nodes to crash. ([Bug#26997](#))
- `mysqld` processes would sometimes crash under high load. ([Bug#26825](#))

- **Disk Data:** When creating a log file group, setting `INITIAL_SIZE` to less than `UNDO_BUFFER_SIZE` caused data nodes to crash. (Bug#25743)

C.1.14.16. Changes in release MySQL 5.1.15-ndb-6.1.4-beta (09 March 2007 - testing only)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

This was a testing release only, and not intended for wide distribution.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, and MySQL 5.1.15-ndb-6.1.3. This version also incorporates all bugfixes and feature changes which were added in the mainline MySQL 5.1 releases up to and including 5.1.15 (see [Section C.1.13](#), “Changes in release 5.1.15 (25 January 2007)”).

Important

Upgrading to MySQL 5.1.15-ndb-6.1.4 from a previous release. This release is not binary compatible with previous releases. This means that:

- You cannot perform an online upgrade to this release from MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, MySQL 5.1.15-ndb-6.1.2, or any of the mainline MySQL 5.1 releases. When upgrading from one of these versions, you must shut down the cluster, replace all binaries, then restart the cluster.
- You must recompile all NDB API and MGM API applications used with a previous version of MySQL Cluster, including those compiled against MySQL 5.1.15-ndb-6.1.3.

Functionality added or changed:

- **NDB API:** It is now possible to specify the transaction coordinator when starting a transaction. See `Ndb::startTransaction()`, for more information.
- **NDB API:** It is now possible to iterate over all existing `Ndb` objects using three new methods of the `Ndb_cluster_connection` class: `lock_ndb_objects()`, `get_next_ndb_object()`, and `unlock_ndb_objects()`. For more information about these methods and their use, see `ndb_cluster_connection::get_next_ndb_object()`, in the MySQL Cluster API Guide.
- A new option `--ndb-wait-connected` has been added for `mysqld`. It causes `mysqld` wait a specified amount of time to be connected to the cluster before starting to accept client connections. See [here](#), for more information.
- Data node memory allocation has been improved. On 32-bit platforms, it should now be possible to use close to 3GB RAM for `In-dexMemory` and `DataMemory` combined.

This release fixes the following bugs:

- Using only the `--print_data` option (and no other options) with `ndb_restore` caused `ndb_restore` to fail. (Bug#26741)
This bug was introduced by the fix for Bug#14612.
- An inadvertent use of unaligned data caused `ndb_restore` to fail on some 64-bit platforms, including Sparc and Itanium-2. (Bug#26739)
- Assigning a node ID greater than 63 to an SQL node caused an out of bounds error in `mysqld`. It should now be possible to assign to SQL nodes node IDs up to 255. (Bug#26663)

C.1.14.17. Changes in release MySQL 5.1.15-ndb-6.1.3-beta (25 February 2007)

This is a new Beta development release, fixing recently discovered bugs and incorporating improvements made since the previous MySQL Cluster 5.1 Carrier Grade Edition release.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.3. The file `mysqlcom-5.1.15-ndb-6.1.3-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, and MySQL 5.1.15-ndb-6.1.2. This version also incorporates all bugfixes and feature changes which were added in the mainline 5.1.15 release; information about these can be found in [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#).

Important

Upgrading to MySQL 5.1.15-ndb-6.1.3 from a previous release. This release is not binary compatible with releases previous to MySQL 5.1.15-ndb-6.1.2. This means that:

- You cannot perform an online upgrade to this release from MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, or any of the mainline MySQL 5.1 releases. When upgrading from one of these versions, you must shut down the cluster, replace all binaries, then restart the cluster.

However, you *can* perform an online upgrade of a Cluster running MySQL 5.1.15-ndb-6.1.2 to MySQL 5.1.15-ndb-6.1.3, using the rolling upgrade procedure described in [Section 16.5.1, “Performing a Rolling Restart of the Cluster”](#). You should replace all binaries for all data nodes, management nodes, and SQL nodes as part of this process. When upgrading the data nodes, you will need to restart each data with the `--initial` option.

- You must recompile all NDB API and MGM API applications used with a previous version of MySQL Cluster, including those compiled against MySQL 5.1.15-ndb-6.1.2.

Functionality added or changed:

- The `ndb_show_tables` utility now displays information about table events. (See [Section 16.9.14, “ndb_show_tables — Display List of NDB Tables”](#).)
- The `ndbd_redo_log_reader` utility is now part of the default build. For more information, see [Section 16.9.11, “ndbd_redo_log_reader — Check and Print Content of Cluster Redo Log”](#).
- It is now possible to disable arbitration by setting `ArbitrationRank=0` on all management and SQL nodes.
- **NDB API:** A new `listEvents()` method has been added to the `Dictionary` class. See [Dictionary Class Methods](#), for more information.

This release fixes the following bugs:

- **NDB Cluster:** An invalid pointer was returned following a `FSCLOSECONF` signal when accessing the REDO logs during a node restart or system restart. ([Bug#26515](#))
- No appropriate error message was provided when there was insufficient REDO log file space for the cluster to start. ([Bug#25801](#))
- **Disk Data:** Use of a tablespace whose `INITIAL_SIZE` was greater than 1 GB could cause the cluster to crash. ([Bug#26487](#))
- A memory allocation failure in the cluster Subscription Manager could cause the cluster to crash. ([Bug#25239](#))
- The `InvalidUndoBufferSize` error used the same error code (763) as the `IncompatibleVersions` error. `InvalidUndoBufferSize` now uses its own error code (779). ([Bug#26490](#))

- **Disk Data:** A memory overflow could occur with tables having a large amount of data stored on disk, or with queries using a very high degree of parallelism on Disk Data tables. ([Bug#26514](#))
- The failure of a data node when restarting it with `--initial` could lead to failures of subsequent data node restarts. ([Bug#26481](#))
- Takeover for local checkpointing due to multiple failures of master nodes was sometimes handled incorrectly. ([Bug#26457](#))
- The `LockPagesInMemory` parameter was not read until after distributed communication had already started between cluster nodes. When the value of this parameter was `1`, this could sometimes result in data node failure due to missed heartbeats. ([Bug#26454](#))
- Under some circumstances, following the restart of a management node, all cluster data nodes would connect to it normally, but some of them subsequently failed to log any events to the management node. ([Bug#26293](#))
- An error was produced when `SHOW TABLE STATUS` was used on an NDB table that had no `AUTO_INCREMENT` column. ([Bug#21033](#))

Note

This improves on and supersedes an earlier fix that was made for this issue in MySQL 5.1.12.

C.1.14.18. Changes in release MySQL 5.1.15-ndb-6.1.2-beta (07 February 2007)

This is a new Beta development release, fixing a recently discovered bug.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.2. The file `mysqlcom-5.1.15-ndb-6.1.2-telco.tar.gz` in this directory contains the complete source archive.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.2. The file `mysqlcom-5.1.15-ndb-6.1.2-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This Beta release is a hotfix release, made to resolve a single critical issue which occurred in MySQL 5.1.15-ndb-6.1.1-beta. It incorporates all bugfixes and feature changes made in MySQL 5.1.14-ndb-6.1.0 and MySQL 5.1.15-ndb-6.1.1. This version also incorporates all bugfixes and feature changes which were added in the mainline 5.1.15 release; information about these can be found in [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#).

Important

Upgrading to MySQL 5.1.15-ndb-6.1.2 from a previous release. This release is not binary compatible with any previous releases. This means that:

- You cannot perform an online upgrade to this release from MySQL 5.1.14-ndb-6.1.0, MySQL 5.1.15-ndb-6.1.1, or any of the mainline MySQL 5.1 releases. You must shut down the cluster, replace all binaries, then restart the cluster.
- You must recompile any MySQL Cluster API applications used with a previous version of MySQL Cluster. This includes all NDB API and MGM API applications.

This release fixes a single critical bug:

- Using node IDs greater than 48 could sometimes lead to incorrect memory access and a subsequent forced shutdown of the cluster. ([Bug#26267](#))

C.1.14.19. Changes in release MySQL 5.1.15-ndb-6.1.1-beta (01 February 2007)

This is a new Beta development release, fixing recently discovered bugs.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.15-ndb-6.1.1. The file `mysqlcom-5.1.15-ndb-6.1.1-telco.tar.gz` in this directory contains the complete source archive.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This section documents all changes and bug fixes that have been applied in this Beta release since the release of MySQL Cluster 5.1.14-ndb-6.1.0. This version also incorporates all bugfixes and feature changes which were added in the mainline 5.1.15 release; information about these can be found in [Section C.1.13, “Changes in release 5.1.15 \(25 January 2007\)”](#).

Important

Upgrading to MySQL 5.1.15-ndb-6.1.1 from a previous release. This release is not binary compatible with any previous releases. This means that:

- You cannot perform an online upgrade to this release from either MySQL 5.1.14-ndb-6.1.0 or from any of the mainline MySQL 5.1 releases. You must shut down the cluster, replace all binaries, then restart the cluster.
- You must recompile any MySQL Cluster API applications used with a previous version of MySQL Cluster. This includes all NDB API and MGM API applications.

Functionality added or changed:

- A single cluster can now support up to 255 API nodes, including MySQL servers acting as SQL nodes. See [Issues exclusive to MySQL Cluster](#) for more information.
- The `LockPagesInMainMemory` configuration parameter has changed its type and possible values. For more information, see [LockPagesInMainMemory](#). (Bug#25686)

Important

The values `true` and `false` are no longer accepted for this parameter. If you were using this parameter and had it set to `false` in a previous release, you must change it to `0`. If you had this parameter set to `true`, you should instead use `1` to obtain the same behavior as previously, or `2` to take advantage of new functionality introduced with this release described in the section cited above.

Bugs fixed:

- A memory leak could cause problems during a node or cluster shutdown or failure. (Bug#25997)
- **Cluster Replication:** Certain errors in replication setups could lead to subsequent node failures. (Bug#25755)
- Hosts in clusters with a large number of nodes could experience excessive CPU usage while obtaining configuration data. (Bug#25711)
- When stopping and restarting multiple data nodes, the last node to be restarted would sometimes hang in Phase 100. (Bug#19645)
- **Cluster APIs / Disk Data:** A delete and a read performed in the same operation could cause one or more of the cluster's data nodes to crash. This could occur when the operation affected more than 5 columns concurrently, or when one or more of the columns was of the VARCHAR type and was stored on disk. (Bug#25794)
- An element could sometimes be inserted twice into the hash table, causing a data node to crash. (Bug#25286)

C.1.15. Changes in release 5.1.14 (05 December 2006)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- **Incompatible change:** Previously, you could create a user-defined function (UDF) or stored function with the same name as a built-in function, but could not invoke the UDF. Now an error occurs if you try to create such a UDF. The server also now generates a warning if you create a stored function with the same name as a built-in function. It is not considered an error to create a stored function with the same name as a built-in function because you can invoke the function using `db_name.func_name()` syntax. However, the server now generates a warning in this case. (Bug#22619, Bug#18239)

See [Section 8.2.4, “Function Name Parsing and Resolution”](#), for the rules describing how the server interprets references to different kinds of functions.

- **Incompatible change:** The `prepared_stmt_count` system variable has been converted to the `Prepared_stmt_count` global status variable (viewable with the `SHOW GLOBAL STATUS` statement). (Bug#23159)
- **NDB Cluster:** Two major changes have taken place with regard to the MySQL Cluster system tables. These are:
 1. **Incompatible change:** The `cluster` database is no longer used. The tables formerly found in the `cluster` database are now in the `mysql` database, and have been renamed as `ndb_binlog_index`, `ndb_apply_status`, and `ndb_schema`.
 2. The `mysql.ndb_apply_status` and `mysql.ndb_schema` tables (formerly `cluster.apply_status` and `cluster.schema` are now created by `ndb_restore`, in the event that they do not already exist on the slave cluster. (Bug#14612)

Note

When upgrading from versions of MySQL previous to 5.1.14 to 5.1.14 or later, `mysql_fix_privilege_tables` merely creates a new `mysql.ndb_binlog_index` table, but does not remove the existing `cluster` database (or, if upgrading from MySQL 5.1.7 or earlier, the existing `cluster_replication` database), nor any of the tables in it.

For more information, see [Section 16.10.4, “Cluster Replication Schema and Tables”](#).

- **NDB Cluster:** It is now possible to create a unique hashed index on a column that is not defined as `NOT NULL`. *Note that this change applies only to tables using the NDB storage engine.*

Unique indexes on columns in NDB tables do not store null values because they are mapped to primary keys in an internal index table (and primary keys cannot contain nulls).

Normally, an additional ordered index is created when one creates unique indexes on NDB table columns; this can be used to search for NULL values. However, if `USING HASH` is specified when such an index is created, no ordered index is created.

The reason for permitting unique hash indexes with null values is that, in some cases, the user wants to save space if a large number of records are pre-allocated but not fully initialized. This also assumes that the user will *not* try to search for null values. Since MySQL does not support indexes that are not allowed to be searched in some cases, the NDB storage engine uses a full table scan with pushed conditions for the referenced index columns to return the correct result.

Note that a warning is returned if one creates a unique nullable hash index, since the query optimizer should be provided a hint not to use it with NULL values if this can be avoided.

(Bug#21507)

- **NDB Cluster**: Backup messages are now printed to the Cluster log. (Bug#24544)
- **NDB Cluster**: The error message `MANAGEMENT SERVER CLOSED CONNECTION`, when recorded in the MySQL error log, now includes a timestamp indicating when the error took place. (Bug#21519)
- **NDB Cluster** (Disk Data): The output of `mysqldump` now includes by default all tablespace and logfile group definitions used by any tables or databases that are dumped. (Bug#20839)

Note

The working of the `--all-tablespaces` or `-Y` option for `mysqldump` remains unaffected by this change.

- Direct and indirect usage of stored routines, user-defined functions, and table references is now prohibited in `CREATE EVENT` and `ALTER EVENT` statements. (Bug#22830)

See [Section 21.2.2, “CREATE EVENT Syntax”](#), and [Section 21.2.1, “ALTER EVENT Syntax”](#), for more specific information.

- `DROP TRIGGER` now supports an `IF EXISTS` clause. (Bug#23703)
- All standard XPath comparison operators are now supported for use with the `ExtractValue()` and `UpdateXML()` functions. (Bug#22823)
- For the `mysql` client, display of result set metadata now is enabled with the `--column-type-info` option rather than with `-debug-info/-T`.

Bugs fixed:

- **NDB Cluster**: If the value set for `MaxNoOfAttributes` is excessive, a suitable error message is now returned. (Bug#19352)
- **NDB Cluster**: Setting the configuration parameter `LockPagesInMainMemory` had no effect. (Bug#24461)
- **NDB Cluster**: Multiple occurrences of error conditions were logged with duplicated error messages rather than being reported with a single error message stating that the error was encountered *N* times. (Bug#22313)
- **NDB Cluster**: Sudden disconnection of an SQL or data node could lead to shutdown of data nodes with the error `FAILED NDBREQUIRE`. (Bug#24447)
- **NDB Cluster**: Different error messages were returned for similar cases involving failure to allocate memory for Cluster operations. (Bug#19203)
- **NDB Cluster**: Some values of `MaxNoOfTriggers` could cause the server to become inaccessible following startup of the data nodes. (Bug#19454)
- **NDB Cluster** (Replication): If errors occurred during purging of the binary logs, extraneous rows could remain left in the `bin-log_index` table. (Bug#15021)
- **NDB Cluster** (Disk Data): `ndb_restore` could sometimes fail when attempting to restore Disk Data tables due to data node failure caused by accessing uninitialized memory. (Bug#24331)
- **NDB Cluster** (Disk Data): Excessive fragmentation of Disk Data files (including log files and data files) could occur during the course of normal use. (Bug#24143)
- **NDB Cluster** (Disk Data): It was possible to execute a statement for creating a Disk Data table that referred to a nonexistent tablespace, in which case the table was an in-memory **NDB** table. Such a statement instead now fails with an appropriate error message. (Bug#23576)
- **NDB Cluster** (Disk Data): Under some circumstances, a `DELETE` from a Disk Data table could cause `mysqld` to crash. (Bug#23542)
- **NDB Cluster** (Cluster APIs): Using `BIT` values with any of the comparison methods of the `NdbScanFilter` class caused the cluster's data nodes to fail. (Bug#24503)
- **NDB Cluster**: A value equal to or greater than the allowed maximum for `LongMessageBuffer` caused all data nodes to crash. (Bug#22547)

- **NDB Cluster**: The failure of a data node failure during a schema operation could lead to additional node failures. (Bug#24752)
- **NDB Cluster**: A committed read could be attempted before a data node had time to connect, causing a timeout error. (Bug#24717)
- **NDB Cluster**: The simultaneous shutdown of `mysqld` and `ndbd` processes caused unnecessary locking. (Bug#24655)
- **NDB Cluster**: The failure of the master node in a node group during the allocation of node IDs could cause `ndb_mgmd` to hang. (Bug#24543)
- **NDB Cluster**: In certain rare cases, a data node could crash due to a typographical error in the MySQL Cluster source code. (Bug#24476)
- **NDB Cluster**: Creating a new tables containing a **BLOB** column when the server was short of memory could cause the server to crash. (Bug#24470)
- **NDB Cluster**: Any statement following the execution of `CREATE TABLE ... LIKE ndb_table` (where `ndb_table` was a table using the **NDB** storage engine), would cause the `mysql` client to hang. (Bug#24301)
- **NDB Cluster**: When the management client command `ALL RESTART -i` was executed while one data node was not running, all data nodes in the cluster were shut down. (Bug#24105)
- **NDB Cluster**: A query using an index scan followed by a delete operation, and then a rollback could cause one or more data nodes to crash. (Bug#24039)
- **NDB Cluster** (Cluster APIs): Some MGM API function calls could yield incorrect return values in certain cases where the cluster was operating under a very high load, or experienced timeouts in inter-node communications. (Bug#24011)
- **NDB Cluster**: It was possible for the sum of the `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUnique-HashIndexes` configuration parameters, plus the number of system tables to exceed the maximum value for a `UInt32` number. In such a case, the cluster's data nodes failed to start, and no reason for this could easily be determined from the error messages provided. (Bug#22548)
- **NDB Cluster**: Given a table `mytbl` in a database `mydb` on a MySQL Server acting as an SQL node in a MySQL Cluster, then, following multiple `ALTER TABLE mytbl ENGINE=engine` statements — first, to change the storage engine used for a table to **NDB**, and then again to change the table to use a non-**NDB** storage engine — a `DROP DATABASE mydb` statement executed on any SQL node in the cluster would cause `mydb` to be dropped on *all* SQL nodes in the cluster, even if `mydb` contained non-**NDB** tables. (Bug#21495)
- **NDB Cluster**: An incorrect error message was displayed in the event that the value of the `MaxNoOfOrderedIndexes` parameter was set too low. (Bug#20065)
- **NDB Cluster**: An incorrect error message was displayed in the event that the value of the `DataMemory` parameter was insufficient for the amount of data to be stored by the cluster. (Bug#19808)
- **NDB Cluster**: A unique constraint violation was not ignored by an `UPDATE IGNORE` statement when the constraint violation occurred on a non-primary key. (Bug#18487, Bug#24303)
- The stack size for NetWare binaries was increased to 128KB to prevent problems caused by insufficient stack size. (Bug#23504)
- Attempting to use a view containing `DEFINER` information for a non-existent user resulted in an error message that revealed the definer account. Now the definer is revealed only to superusers. Other users receive only an `access denied` message. (Bug#17254)
- The size of **MEMORY** tables and internal temporary tables was limited to 4GB on 64-bit Windows systems. (Bug#24052)
- For debug builds, `mysqladmin shutdown` displayed an extraneous `skipped 9 bytes from file: socket (3)` message. (Bug#21428)
- For queries that select from a view, the server was returning `MYSQL_FIELD` metadata inconsistently for view names and table names. For view columns, the server now returns the view name in the `table` field and, if the column selects from an underlying table, the table name in the `org_table` field. (Bug#20191)
- `CREATE FUNCTION X()` and `CREATE FUNCTION Y()` failed with a syntax error instead of warning the user that these function names are already used (for GIS functions). (Bug#21025)

- The loose index scan optimization for `GROUP BY` with `MIN` or `MAX` was not applied within other queries, such as `CREATE TABLE ... SELECT ...`, `INSERT ... SELECT ...`, or in the `FROM` clauses of subqueries. (Bug#24156)
- Queries using a column alias in an expression as part of an `ORDER BY` clause failed, an example of such a query being `SELECT mycol + 1 AS mynum FROM mytable ORDER BY 30 - mynum`. (Bug#22457)
- Trailing spaces were not removed from Unicode `CHAR` column values when used in indexes. This resulted in excessive usage of storage space, and could affect the results of some `ORDER BY` queries that made use of such indexes.

Note

When upgrading, it is necessary to re-create any existing indexes on Unicode `CHAR` columns in order to take advantage of the fix. This can be done by using a `REPAIR TABLE` statement on each affected table.

(Bug#22052)

- Warnings were generated when explicitly casting a character to a number (for example, `CAST('x' AS SIGNED)`), but not for implicit conversions in simple arithmetic operations (such as `'x' + 0`). Now warnings are generated in all cases. (Bug#11927)
- `BENCHMARK`, `ENCODE`, `DECODE`, and `FORMAT` could only accept a constant for some parameters, and could not be used in prepared statements. (Bug#22684)
- A query with a subquery that references columns of a view from the outer `SELECT` could return an incorrect result if used from a prepared statement. (Bug#20327)
- Constant expressions and some numeric constants used as input parameters to user-defined functions were not treated as constants. (Bug#18761)
- `INET_ATON()` returned a signed `BIGINT` value, not an unsigned value. (Bug#21466)
- In some cases, the parser failed to distinguish a user-defined function from a stored function. (Bug#21809)
- In some cases, a function that should be parsed as a user-defined function was parsed as a stored function. (Bug#24736)
- Subqueries for which a pushed-down condition did not produce exactly one key field could cause a server crash. (Bug#24056)
- `LAST_DAY('0000-00-00')` could cause a server crash. (Bug#23653)
- Through the C API, the member strings in `MYSQL_FIELD` for a query that contains expressions may return incorrect results. (Bug#21635)
- `mysql_affected_rows()` could return values different from `mysql_stmt_affected_rows()` for the same sequence of statements. (Bug#23383)
- `IN()` and `CHAR()` can return `NULL`, but did not signal that to the query processor, causing incorrect results for `IS NULL` operations. (Bug#17047)
- Instance Manager option-parsing code caused memory-allocation errors. (Bug#22242)
- A trigger that invoked a stored function could cause a server crash when activated by different client connections. (Bug#23651)
- `CONCURRENT` did not work correctly for `LOAD DATA INFILE`. (Bug#20637)
- The server source code had multiple exportable definitions of the `field_in_record_is_null()` function. These are now all declared `static`. (Bug#24190)
- Inserting a default or invalid value into a spatial column could fail with `Unknown error` rather than a more appropriate error. (Bug#21790)
- The server could send incorrect column count information to the client for queries that produce a larger number of columns than can fit in a two-byte number. (Bug#19216)
- Evaluation of subqueries that require the filesort algorithm were allocating and freeing the `sort_buffer_size` buffer many times, resulting in slow performance. Now the buffer is allocated once and reused. (Bug#21727)
- SQL statements close to the size of `max_allowed_packet` could produce binary log events larger than

`max_allowed_packet` that could not be read by slave servers. (Bug#19402)

- View columns were always handled as having implicit derivation, leading to `illegal mix of collation errors` for some views in `UNION` operations. Now view column derivation comes from the original expression given in the view definition. (Bug#21505)
- If elements in a non-top-level `IN` subquery were accessed by an index and the subquery result set included a `NULL` value, the quantified predicate that contained the subquery was evaluated to `NULL` when it should return a non-`NULL` value. (Bug#23478)
- Calculation of `COUNT(DISTINCT)`, `AVG(DISTINCT)`, or `SUM(DISTINCT)` when they are referenced more than once in a single query with `GROUP BY` could cause a server crash. (Bug#23184)
- For a cast of a `DATETIME` value containing microseconds to `DECIMAL`, the microseconds part was truncated without generating a warning. Now the microseconds part is preserved. (Bug#19491)
- Metadata for columns calculated from scalar subqueries was limited to integer, double, or string, even if the actual type of the column was different. (Bug#11032)
- Some unnecessary Valgrind warnings were removed from the server. (Bug#24488, Bug#24533).
- Using `EXPLAIN` caused a server crash for queries that selected from `INFORMATION_SCHEMA` in a subquery in the `FROM` clause. (Bug#22413)
- Invalidating the query cache caused a server crash for `INSERT INTO ... SELECT` statements that selected from a view. (Bug#20045)
- With row-based binary logging, replicated multiple-statement transaction deadlocks did not return the correct error code, causing the slave SQL thread to stop rather than roll back and re-execute. (Bug#23831)
- With row-based binary logging, for `CREATE TABLE IF NOT EXISTS LIKE temporary_table` statements, the `IF NOT EXISTS` clause was not logged. (Bug#22762)
- With row-based binary logging, `CREATE TABLE IF NOT EXISTS SELECT` statements were not logged properly. (Bug#22027)
- On slave servers, transactions that exceeded the lock wait timeout failed to roll back properly. (Bug#20697)
- Changes to character set variables prior to an action on a replication-ignored table were forgotten by slave servers. (Bug#22877)
- With `lower_case_table_names` set to 1, `SHOW CREATE TABLE` printed incorrect output for table names containing Turkish I (LATIN CAPITAL LETTER I WITH DOT ABOVE). (Bug#20404)
- When applying the `group_concat_max_len` limit, `GROUP_CONCAT()` could truncate multi-byte characters in the middle. (Bug#23451)
- For view renaming, the table name to filename encoding was not performed. (Bug#21370)
- For some problems relating to character set conversion or incorrect string values for `INSERT` or `UPDATE`, the server was reporting truncation or length errors instead. (Bug#18908)
- The XPath operators `<` and `>`, as implemented in the `ExtractValue()` function, operated in reverse. (Bug#22823)

C.1.16. Changes in MySQL 5.1.14 Carrier Grade Edition

This section contains change history information for MySQL Cluster 5.1 Carrier Grade Edition releases based on MySQL 5.1.14.

C.1.16.1. Changes in release MySQL 5.1.14-ndb-6.1.0-beta (20 December 2006)

This is a new Beta development release, fixing recently discovered bugs.

Like all releases for MySQL MySQL Cluster 5.1 Carrier Grade Edition, this is a source-only release which you must compile and install using the instructions found in [Section 2.9, “MySQL Installation Using a Source Distribution”](#), and in [Section 16.4.1, “Building MySQL Cluster from Source Code”](#). You can download the source code archive for this release from the MySQL FTP site at ftp://ftp.mysql.com/pub/mysql/download/cluster_telco/mysql-5.1.14-ndb-6.1.0. There are two version of the source archive in this dir-

ectory:

- [mysql-5.1.14-ndb-6.1.0-telco.tar.gz](#): This is the GPL version. Customers who are using MySQL MySQL Cluster 5.1 Carrier Grade Edition under a commercial licence must *not* use this version in a production setting.
- [mysqlcom-5.1.14-ndb-6.1.0-telco.tar.gz](#): This is the commercial version. Customers who are using MySQL MySQL Cluster 5.1 Carrier Grade Edition under a commercial licence *must* use this version in any setting covered under their licence agreement with MySQL AB.

NOTE: Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This section documents all changes and bug fixes that have been applied in this Beta release since MySQL MySQL Cluster 5.1 Carrier Grade Edition diverged from MySQL 5.1.14 standard.

Functionality added or changed:

- Support was added for a new [MemReportFrequency](#) configuration parameter that provides for periodic reports of memory usage. See [Section 16.4.5.2, “Management Node Configuration Parameters”](#), for information about using this parameter.

Bugs fixed:

- **NDB API:** A unique index lookup on a non-existent tuple could lead to a data node timeout (error 4012). ([Bug#25059](#))
- A [SELECT](#) statement performing a primary key lookup on a table with a [VARCHAR](#) primary key, and where a [BLOB](#) or [TEXT](#) column was among the columns selected, would produce an empty result set. ([Bug#19956](#))
- **NDB API:** When using the [NdbTransaction::execute\(\)](#) method, a very long timeout (greater than 5 minutes) could result if the last data node being polled was disconnected from the cluster. ([Bug#24949](#))
- **Disk Data:** A [MEDIUMTEXT](#) column of a Disk Data table was stored in memory rather than on disk, even if the column was not indexed. ([Bug#25001](#))
- Under certain rare circumstances, local checkpoints were not performed properly, leading to an inability to restart one or more data nodes. ([Bug#24664](#))
- **Disk Data:** Performing a node restart with a newly dropped Disk Data table could lead to failure of the node during the restart. ([Bug#24917](#))
- **Disk Data:** Repeated [CREATE](#), [DROP](#), or [TRUNCATE](#) in various combinations with system restarts between these operations could lead to the eventual failure of a system restart. ([Bug#21948](#))
- **Disk Data:** Extents that should have been available for re-use following a DROP TABLE operation were not actually made available again until after the cluster performed a local checkpoint. ([Bug#17605](#))
- When a data node was shut down using the management client [STOP](#) command, a connection event ([NDB_LE_Connected](#)) was logged instead of a disconnection event ([NDB_LE_Disconnected](#)). ([Bug#22773](#))
- **Disk Data:** When restoring from backup a cluster containing any Disk Data tables with hidden primary keys, a node failure resulted which could lead to a crash of the cluster. ([Bug#24166](#))
- **NDB API:** Invoking the [NdbTransaction::execute\(\)](#) method using execution type [Commit](#) and abort option [AO_IgnoreError](#) could lead to a crash of the transaction coordinator ([DBTC](#)). ([Bug#25090](#))
- **NDB API:** Due to an error in the computation of table fragment arrays, some transactions might not be executed from the correct starting point. ([Bug#24914](#))

C.1.17. Changes in release 5.1.13 (Not released)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- **Incompatible change:** The number of function names affected by `IGNORE_SPACE` was reduced significantly in MySQL 5.1.13, from about 200 to about 30. (For details about `IGNORE_SPACE`, see [Section 8.2.4, “Function Name Parsing and Resolution”](#).) This change improves the consistency of parser operation. However, it also introduces the possibility of incompatibility for old SQL code that relies on the following conditions:

- `IGNORE_SPACE` is disabled.
- The presence or absence of whitespace following a function name is used to distinguish between a built-in function and stored function that have the same name (for example, `PI()` versus `PI ()`).

For functions that are no longer affected by `IGNORE_SPACE` as of MySQL 5.1.13, that strategy no longer works. Either of the following approaches can be used if you have code that is subject to the preceding incompatibility:

- If a stored function has a name that conflicts with a built-in function, refer to the stored function with a schema name qualifier, regardless of whether whitespace is present. For example, write `schema_name.PI()` or `schema_name.PI ()`.
- Alternatively, rename the stored function to use a non-conflicting name and change invocations of the function to use the new name.

([Bug#21114](#))

- If the user specified the server options `--max-connections=N` or `--table-open-cache=M`, a warning would be given in some cases that some values were recalculated, with the result that `--table-open-cache` could be assigned greater value.

It should be noted that, in such cases, both the warning and the increase in the `--table-open-cache` value were completely harmless. Note also that it is not possible for the MySQL Server to predict or to control limitations on the maximum number of open files, since this is determined by the operating system.

The recalculation code has now been fixed to ensure that the value of `--table-open-cache` is no longer increased automatically, and that a warning is now given only if some values had to be decreased due to operating system limits.

([Bug#21915](#))

- Binary distributions of MySQL 5.1.12 were built without support for partitioning. This has been corrected except for NetWare. ([Bug#23949](#))
- A change in the interfaces for the `INFORMATION_SCHEMA.FILES` table has made the table accessible to storage engines other than `NDB`. ([Bug#23013](#))
- `mysqldump --single-transaction` now uses `START TRANSACTION /*!40100 WITH CONSISTENT SNAPSHOT */` rather than `BEGIN` to start a transaction, so that a consistent snapshot will be used on those servers that support it. ([Bug#19660](#))
- `mysql_upgrade` now passes all the parameters specified on the command line to both `mysqlcheck` and `mysql` using the `upgrade_defaults` file. ([Bug#20100](#))
- For the `CALL` statement, stored procedures that take no arguments now can be invoked without parentheses. That is, `CALL p()` and `CALL p` are equivalent. ([Bug#21462](#))

Bugs fixed:

- MySQL 5.0.26 introduced an ABI incompatibility, which this release reverts. Programs compiled against 5.0.26 are not compatible

with any other version and must be recompiled. ([Bug#23427](#))

- Several string functions could return incorrect results when given very large length arguments. ([Bug#10963](#))
- If an `init_connect` SQL statement produced an error, the connection was silently terminated with no error message. Now the server writes a warning to the error log. ([Bug#22158](#))
- If a table contains an `AUTO_INCREMENT` column, inserting into an insertable view on the table that does not include the `AUTO_INCREMENT` column should not change the value of `LAST_INSERT_ID()`, because the side effects of inserting default values into columns not part of the view should not be visible. MySQL was incorrectly setting `LAST_INSERT_ID()` to zero. ([Bug#22584](#))
- `M % 0` returns `NULL`, but `(M % 0) IS NULL` evaluated to false. ([Bug#23411](#))
- Within a stored routine, a view definition cannot refer to routine parameters or local variables. However, an error did not occur until the routine was called. Now it occurs during parsing of the routine creation statement. ([Bug#20953](#))

Note

A side effect of this fix is that if you have already created such routines, an error will occur if you execute `SHOW CREATE PROCEDURE` or `SHOW CREATE FUNCTION`. You should drop these routines because they are erroneous.

- A client library crash was caused by executing a statement such as `SELECT * FROM t1 PROCEDURE ANALYSE()` using a server side cursor on a table `t1` that does not have the same number of columns as the output from `PROCEDURE ANALYSE()`. ([Bug#17039](#))
- `mysql` did not check for errors when fetching data during result set printing. ([Bug#22913](#))
- Adding a day, month, or year interval to a `DATE` value produced a `DATE`, but adding a week interval produced a `DATETIME` value. Now all produce a `DATE` value. ([Bug#21811](#))
- The column default value in the output from `SHOW COLUMNS` or `SELECT FROM INFORMATION_SCHEMA.COLUMNS` was truncated to 64 characters. ([Bug#23037](#))
- For not-yet-authenticated connections, the `Time` column in `SHOW PROCESSLIST` was a random value rather than `NULL`. ([Bug#23379](#))
- The `Host` column in `SHOW PROCESSLIST` output was blank when the server was started with the `--skip-grant-tables` option. ([Bug#22728](#))
- The `Handler_rollback` status variable sometimes was incremented when no rollback had taken place. ([Bug#22728](#))
- Within a prepared statement, `SELECT (COUNT(*) = 1)` (or similar use of other aggregate functions) did not return the correct result for statement re-execution. ([Bug#21354](#))
- Lack of validation for input and output `TIME` values resulted in several problems: `SEC_TO_TIME()` within subqueries incorrectly clipped large values; `SEC_TO_TIME()` treated `BIGINT UNSIGNED` values as signed; only truncation warnings were produced when both truncation and out-of-range `TIME` values occurred. ([Bug#11655](#), [Bug#20927](#))
- Range searches on columns with an index prefix could miss records. ([Bug#20732](#))
- With `SQL_MODE=TRADITIONAL`, MySQL incorrectly aborted on warnings within stored routines and triggers. ([Bug#20028](#))
- In `mysql`, invoking `connect` or `\r` with very long `db_name` or `host_name` parameters caused buffer overflow. ([Bug#20894](#))
- `mysqldump --xml` produced invalid XML for `BLOB` data. ([Bug#19745](#))
- For a debug server, a reference to an undefined user variable in a prepared statement executed with `EXECUTE` caused an assertion failure. ([Bug#19356](#))
- Within a trigger for a base table, selecting from a view on that base table failed. ([Bug#19111](#))
- `DELETE IGNORE` could hang for foreign key parent deletes. ([Bug#18819](#))
- Transient errors in replication from master to slave may trigger multiple `Got fatal error 1236: 'binlog truncated in the middle of event'` errors on the slave. ([Bug#4053](#))

- The value of the `warning_count` system variable was not being calculated correctly (also affecting `SHOW COUNT(*) WARNINGS`). (Bug#19024)
- `InnoDB` showed substandard performance with multiple queries running concurrently. (Bug#15815)
- There was a race condition in the `InnoDB fil_flush_file_spaces()` function. (Bug#24098)
- `FROM_UNIXTIME()` did not accept arguments up to `POWER(2, 31) - 1`, which it had previously. (Bug#9191)
- Some yaSSL-related memory leaks detected by Valgrind were fixed. (Bug#23981)
- If `COMPRESS()` returned `NULL`, subsequent invocations of `COMPRESS()` within a result set or within a trigger also returned `NULL`. (Bug#23254)
- `mysql` would lose its connection to the server if its standard output was not writable. (Bug#17583)
- `mysql-test-run` did not work correctly for RPM-based installations. (Bug#17194)
- The return value from `my_seek()` was ignored. (Bug#22828)
- Use of `PREPARE` with a `CREATE PROCEDURE` statement that contained a syntax error caused a server crash. (Bug#21868)
- Use of a DES-encrypted SSL certificate file caused a server crash. (Bug#21868)
- Column names were not quoted properly for replicated views. (Bug#19736)
- `InnoDB` used table locks (not row locks) within stored functions. (Bug#18077)
- `InnoDB` crashed when trying to display an error message about a foreign key constraint violation when the two tables are in different schemas. (Bug#23368)
- Statements such as `DROP PROCEDURE` and `DROP VIEW` were written to the binary log too late due to a race condition. (Bug#14262)
- At shutdown, Instance Manager told guarded server instances to stop, but did not wait until they actually stopped. (Bug#17486)
- It was not possible to do an atomic rename of the log tables without the possibility of losing rows. Now you can do this:

```
USE mysql;
CREATE TABLE IF NOT EXISTS general_log2 LIKE general_log;
RENAME TABLE general_log TO general_log_backup, general_log2 TO general_log;
```

(Bug#17544, Bug#21785)

- **NDB Cluster** (Disk Data): In the event of an aborted multiple update, the space in the Disk Data log buffer to be freed as a result was actually freed twice, which could eventually lead to a crash. (Bug#23430)
 - **NDB Cluster**: An `NDB` source file included a `memset()` call with reversed arguments. (Bug#23169)
 - **NDB Cluster**: Under some conditions, the data distribution could become unbalanced in a MySQL Cluster with 2 or more node groups following the creation of a new table. (Bug#21690)
- As part of the fix for this bug, two new `NDB` API methods were added to the `NdbDictionary::Object::Table` class. See the MySQL Cluster API documentation for details.
- Incorrect warnings occurred for use of `CREATE TABLE ... LIKE` or `REPAIR TABLE` with the log tables. (Bug#21966)
 - MySQL failed to build on the Alpha platform. (Bug#23256)
 - The optimizer failed to use equality propagation for `BETWEEN` and `IN` predicates with string arguments. (Bug#22753)
 - The optimizer used the `ref` join type rather than `eq_ref` for a simple join on strings. (Bug#22367)
 - The `WITH CHECK OPTION` for a view failed to prevent storing invalid column values for `UPDATE` statements. (Bug#16813)
 - A literal string in a `GROUP BY` clause could be interpreted as a column name. (Bug#14019)

- Some queries that used `MAX()` and `GROUP BY` could incorrectly return an empty result. (Bug#22342)
- `WITH ROLLUP` could group unequal values. (Bug#20825)
- Use of a subquery that invoked a function in the column list of the outer query resulted in a memory leak. (Bug#21798)
- `LIKE` searches failed for indexed `utf8` character columns. (Bug#20471)
- `FLUSH INSTANCES` in Instance Manager triggered an assertion failure. (Bug#19368)
- `ALTER TABLE` was not able to rename a view. (Bug#14959)
- The optimizer sometimes mishandled R-tree indexes for `GEOMETRY` data types, resulting in a server crash. (Bug#21888)
- An unhandled `NULL` pointer caused a server crash. (Bug#22138)
- Use of `SQL_BIG_RESULT` did not influence the sort plan for query execution. (Bug#22781)
- The server did not allocate sufficient memory for some queries for which a `DISTINCT` to `GROUP BY` conversion is possible and an `ORDER BY` clause is present, resulting in a server crash. (Bug#20503)
- The range analysis optimizer did not take into account predicates for which an index could be used after reading `const` tables. In some cases this resulted in non-optimal execution plans. (Bug#19579)
- Entries in the slow query log could have an incorrect `Rows_examined` value. (Bug#12240)
- Insufficient memory (`myisam_sort_buffer_size`) could cause a server crash for several operations on `MyISAM` tables: repair table, create index by sort, repair by sort, parallel repair, bulk insert. (Bug#23175)
- `OPTIMIZE TABLE` with `myisam_repair_threads > 1` could result in `MyISAM` table corruption. (Bug#8283)
- `NDB Cluster`: Restoring a cluster failed if there were any tables with 128 or more columns. (Bug#23494, Bug#23502)
- `NDB Cluster`: Cluster backups would fail when there were more than 2048 schema objects in the cluster. (Bug#23499)
- `NDB Cluster`: The management client command `ALL DUMP 1000` would cause the cluster to crash if data nodes were connected to the cluster but not yet fully started. (Bug#23203)
- `NDB Cluster`: `INSERT ... ON DUPLICATE KEY UPDATE` on an `NDB` table could lead to deadlocks and memory leaks. (Bug#23200)
- `NDB Cluster`: If a node restart could not be performed from the REDO log, no node takeover took place. This could cause partitions to be left empty during a system restart. (Bug#22893)
- `NDB Cluster`: Multiple node restarts in rapid succession could cause a system restart to fail (Bug#22892), or induce a race condition (Bug#23210).
- `NDB Cluster`: Attempting to create a unique constraint with `USING HASH` on an `NDB` table caused `mysqld` to crash. (Bug#21873)
- `NDB Cluster`: When inserting a row into an `NDB` table with a duplicate value for a non-primary unique key, the error issued would reference the wrong key. (Bug#21072)
- `NDB Cluster` (`NDB API`): When multiple processes or threads in parallel performed the same ordered scan with exclusive lock and updating the retrieved records, the scan could skip some records, which were not updated as the result. (Bug#20446)
- `NDB Cluster`: Aborting a cluster backup too soon after starting it caused a forced shutdown of the data nodes. (Bug#19148)

C.1.18. Changes in release 5.1.12 (24 October 2006)

This is a new Beta development release, fixing recently discovered bugs.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see [ht-](#)

[tp://www.mysql.com/products/enterprise](http://www.mysql.com/products/enterprise).

Functionality added or changed:

- **Incompatible change:** Support for the BerkeleyDB (BDB) engine has been dropped from this release. Any existing tables that are in BDB format will not be readable from within MySQL from 5.1.12 or newer. You should convert your tables to another storage engine *before* upgrading to 5.1.12.

Because of this change, the `SHOW [BDB] LOGS` statement has been dropped.

- **Incompatible change:** The namespace for scheduled events has changed, such that events are no longer unique to individual users. This also means that a user with the `EVENT` privilege on a given database can now view, alter, or drop any events defined on that database.

If you used scheduled events in an earlier MySQL 5.1 release, you should rename any of them having the same name and defined on the same database but belonging to different users — so that all events in a given database have unique names — *before* upgrading to 5.1.12 (or newer).

For additional information, see [Section 21.5, “The Event Scheduler and MySQL Privileges”](#).

- **Incompatible change:** The permitted values for and behaviour of the `event_scheduler` system variable have changed. Permitted values are now `ON`, `OFF`, and `DISABLED`, with `OFF` being the default. It is not possible to change its value to or from `DISABLED` while the server is running.

For details, see [Section 21.1, “Event Scheduler Overview”](#).

- **Incompatible change:** The plugin interface has changed: The `st_mysql_plugin` structure has a new `license` member to indicate the license type. (The allowable values are defined in `mysql/plugin.h`.) This change is not backward compatible, so the API version (`MYSQL_PLUGIN_INTERFACE_VERSION`) has changed. For additional information, see [Section 28.2.5, “Writing Plugins”](#).

- **Incompatible change:** The full-text parser plugin interface has changed in two ways:

- The `MYSQL_FTPARSER_PARAM` structure has a new `flags` member. This is zero if there are no special flags, or `MYSQL_FTFLAGS_NEED_COPY`, which means that `mysql_add_word()` must save a copy of the word (that is, it cannot use a pointer to the word because the word is in a buffer that will be overwritten.)

This flag might be set or reset by MySQL before calling the parser plugin, by the parser plugin itself, or by the `mysql_parse()` function.

- The `mysql_parse()` and `mysql_add_word()` functions now take a `MYSQL_FTPARSER_PARAM` as their first argument, not a `MYSQL_FTPARSER_PARAM : mysql_ftparam` as before.

These changes are not backward compatible, so the API version (`MYSQL_FTPARSER_INTERFACE_VERSION`) has changed. For additional information, see [Section 28.2.5, “Writing Plugins”](#).

- **Incompatible change:** Storage engines can be pluggable at runtime, so the distinction between disabled and invalid storage engines no longer applies. This affects the `NO_ENGINE_SUBSTITUTION` SQL mode, as described in [Section 5.2.6, “SQL Modes”](#).
- **Incompatible change:** In the `INFORMATION_SCHEMA.EVENTS` table, the `EVENT_DEFINITION` column now contains the SQL executed by a scheduled event.

The `EVENT_BODY` column now contains the language used for the statement or statements shown in `EVENT_DEFINITION`. In MySQL 5.1, the value shown in `EVENT_BODY` is always `SQL`.

These changes were made to bring this table into line with the `INFORMATION_SCHEMA.ROUTINES` table, and that table's `ROUTINE_BODY` and `ROUTINE_DEFINITION` columns. ([Bug#16992](#))

- **Incompatible change:** MySQL Cluster node and system restarts formerly required that all fragments use the same local checkpoint (LCP); beginning with this version, it is now possible for different fragments to use different LCPs during restarts. This means that data node filesystems must be rebuilt as part of any upgrade to this version by restarting all data nodes with the `--initial` option. ([Bug#21271](#), [Bug#21478](#))

See [Section 16.5.2, “Cluster Upgrade and Downgrade Compatibility”](#), and related sections of the Manual before upgrading a MySQL Cluster to version 5.1.12 or later.

- **Incompatible change:** A number of MySQL constructs are now prohibited in partitioning expressions, beginning with this release. These include:

- A number of MySQL functions.

You can find [a complete list of these functions](#) under *Partitioning Limitations*.

- The bit operators `|`, `&`, `^`, `<<`, `>>`, and `~`.
- Nested function calls.
- Calls to stored routines, UDFs, or plugins.
- Character-to-integer conversions involving non-8-bit character sets or any of the `latin1_german2_ci`, `latin2_czech_cs`, or `cp1250_czech_cs` collations.

These restrictions were added in part as a result of [Bug#18198](#) and related bug reports.

For more information about these and other restrictions on partitioned tables in MySQL, see [Section 17.5, “Restrictions and Limitations on Partitioning”](#).

- Binary MySQL distributions no longer include a `mysqld-max` server. Instead, distributions contain a binary that includes the features previously included in the `mysqld-max` binary.
- The default binary log format (as used during replication) is now Mixed based, automatically using a combination of row-based and statement based log events as appropriate.
- The general query log and slow query logs now can be enabled or disabled at runtime with the `general_log` and `slow_query_log` system variables, and the name of the log files can be changed by setting the `general_log_file` and `slow_query_log_file` system variables. See [Section 5.11.3, “The General Query Log”](#), and [Section 5.11.5, “The Slow Query Log”](#).
- The output generated by the server when using the `--xml` option has changed with regard to null values. It now matches the output from `mysqldump --xml`. That is, a column containing a `NULL` value is now reported as

```
<field name="column_name" xsi:nil="true" />
```

whereas a column containing the string value `'NULL'` is reported as

```
<field name="column_name">NULL</field>
```

and a column containing an empty string is reported as

```
<field name="column_name">>/field>
```

([Bug#21263](#))

- The following statements now can be executed as prepared statements (using `PREPARE` plus `EXECUTE`):

```
CACHE INDEX
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
{CREATE | RENAME | DROP} DATABASE
{CREATE | RENAME | DROP} USER
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
      | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES}
GRANT
REVOKE
KILL
LOAD INDEX INTO CACHE
RESET {MASTER | SLAVE | QUERY CACHE}
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {AUTHORS | CONTRIBUTORS | WARNINGS | ERRORS}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
INSTALL PLUGIN
UNINSTALL PLUGIN
```

([Bug#20665](#))

- The Instance Manager `--passwd` option has been renamed to `--print-password-line`. Other options were added to enable management of the IM password file from the command line: `--add-user`, `--drop-user`, `--edit-user`, `--list-users`, `--check-password-file`, `--clean-password-file`, `--username`, and `--password`. The `--mysql-safe-compatible` option was added to cause the Instance Manager to act similarly to `mysqld_safe`.
- On Windows, typing Control-C caused the `mysql` client to crash. Now it causes `mysql` to attempt to kill the current statement. If this cannot be done, or Control-C is typed again before the statement is killed, `mysql` exits. ([Bug#17926](#); see also [Bug#1989](#))
- `TEXT` and `BLOB` columns do not support `DEFAULT` values. However, when a default of `' '` was specified, the specification was silently ignored. This now results in a warning, or an error in strict mode. ([Bug#19498](#))
- The `mysql` client now allows `\l` in the `prompt` command argument to insert the current delimiter into the prompt. ([Bug#14448](#))
- Log table changes: By default, the log tables use the `CSV` storage engine, as before. But now the log tables can be altered to use the `MyISAM` storage engine. You cannot use `ALTER TABLE` to alter a log table that is in use. The log must be disabled first. No engines other than `CSV` or `MyISAM` are legal for the log tables. The use of `DROP TABLE` for log tables is similarly restricted: It cannot be used to drop a log table that is in use. The log must be disabled first. (These changes also correct a deadlock that occurred for an attempt to drop an in-use log table.) ([Bug#18559](#))
- The `LEFT()` and `RIGHT()` functions return `NULL` if any argument is `NULL`. ([Bug#11728](#))
- `EXPLAIN EXTENDED` now shows a `filtered` column that is an estimated percentage of the examined rows that will be joined with the previous tables. ([Bug#14940](#))
- For `mysqlshow`, if a database name argument contains wildcard characters (such as `'_'`) but matches a single database name exactly, treat the name as a literal name. This allows a command such as `mysqlshow information_schema` work without having to escape the wildcard character. ([Bug#19147](#))
- If a `DROP VIEW` statement named multiple views, it stopped with an error if a non-existent view was named and did not drop the remaining views. Now it continues on and reports an error at the end, similar to `DROP TABLE`. ([Bug#16614](#))
- `SHOW CREATE TABLE` now shows constraints for `InnoDB` tables. ([Bug#16614](#))
- Added the `--set-charset` option to `mysqlbinlog` to allow the character set to be specified for processing binary log files. ([Bug#18351](#))
- **NDB Cluster:** Backup messages are no longer printed to the Cluster log.
- **NDB Cluster:** The `HELP` command in the Cluster management client now provides command-specific help. For example, `HELP RESTART` in `ndb_mgm` provides detailed information about the `START` command. ([Bug#19620](#))
- **NDB Cluster:** Inserting into an `NDB` table failed when the table had no primary key but had a unique key added after table was created on one or more `NOT NULL` columns. This occurred when the unique key had been adding using either `ALTER TABLE` or `CREATE UNIQUE KEY`. ([Bug#22838](#))
- **NDB Cluster:** The `ndb_config` utility now accepts `-c` as a short form of the `--ndb-connectstring` option. ([Bug#22295](#))
- **NDB Cluster:** Added the `--bind-address` option for `ndbd`. This allows a data node process to be bound to a specific network interface. ([Bug#22195](#))
- **NDB Cluster:** The `Ndb_number_of_storage_nodes` system variable was renamed to `Ndb_number_of_data_nodes`. ([Bug#20848](#))
- Added the `--ndb-use-copying-alter-table` option to `mysqld` to provide a fallback in case of problems with online `ALTER TABLE` operations on `NDBCluster` tables.
- Added the `SHOW CONTRIBUTORS` statement.
- It is no longer possible to create partitioned tables using the `CSV` storage engine.
- **NDB Cluster:** The status variables `Ndb_connected_host` and `Ndb_connected_port` were renamed to `Ndb_config_from_host` and `Ndb_config_from_port`, respectively.

- **NDB Cluster**: A number of erroneous, misleading, or missing error messages have been corrected. ([Bug#17297](#) & [Bug#19543](#))
- **NDB Cluster**: It is no longer possible to create Cluster tables using any partitioning type other than [**LINEAR**] **KEY**. Attempting to do so now raises an error.
- The **ExtractValue()** function now produces an error when passed an XML fragment that is not well-formed. ([Bug#18201](#))
(Previously, the function allowed invalid XML fragments to be used.)
- There were several issues regarding how **SHOW STATUS** affected some status variables and logging which could impact monitoring the MySQL Server. The behavior of this statement has been modified in two ways:
 - **SHOW STATUS** is no longer logged to the slow query log.
 - **SHOW STATUS** no longer updates any session status variables, except for **com_show_status**.

However, **SHOW STATUS** continues to update *global* status variables to allow monitoring of what the server is actually doing. This is because **SHOW STATUS** creates temporary tables that may affect performance if it is called excessively often. ([Bug#10210](#), [Bug#19764](#))

- The **mysqldumps low** script has been moved from client RPM packages to server RPM packages. This corrects a problem where **mysqldumps low** could not be used with a client-only RPM install, because it depends on **my_print_defaults** which is in the server RPM. ([Bug#20216](#))
- The bundled yaSSL library was upgraded to version 1.3.7.
- The bundled yaSSL library licensing has added a FLOSS exception similar to MySQL to resolve licensing incompatibilities with MySQL. (See the [extra/yaSSL/FLOSS-EXCEPTIONS](#) file in a MySQL source distribution for details.) ([Bug#16755](#))
- **mysqlslap** threads now try to connect up to 10 times if the initial connect attempt fails. ([Bug#21297](#))
- For a successful dump, **mysqldump** now writes a SQL comment to the end of the dump file in the following format:

```
-- Dump completed on YYYY-MM-DD hh:mm:ss
```

([Bug#10877](#))

- The **mysqld** and **mysqlmanager** manpages have been reclassified from volume 1 to volume 8. ([Bug#21220](#))
- In the **INFORMATION_SCHEMA.ROUTINES** table the **ROUTINE_DEFINITION** column now is defined as **NULL** rather than **NOT NULL**. Also, **NULL** rather than the empty string is returned as the column value if the user does not have sufficient privileges to see the routine definition. ([Bug#20230](#))
- **configure** now defines the symbol **DEBUG_ON** in **config.h** to indicate whether the source tree is configured to be compiled with debugging support. ([Bug#19517](#))
- The MySQL distribution now compiles on UnixWare 7.13. ([Bug#20190](#))
- The **mysql** client used the default character set if it automatically reconnected to the server, which is incorrect if the character set had been changed. To enable the character set to remain synchronized on the client and server, the **mysql** command **charset** (or **\C**) that changes the default character set and now also issues a **SET NAMES** statement. The changed character set is used for re-connects. ([Bug#11972](#))
- The **STATE** column of the **INFORMATION_SCHEMA.PROCESSLIST** table was increased from 30 to 64 characters to accommodate longer state values. ([Bug#21652](#))
- **TIMESTAMP** columns that are **NOT NULL** now are reported that way by **SHOW COLUMNS** and **INFORMATION_SCHEMA**. ([Bug#20910](#))
- **INFORMATION_SCHEMA** contains new tables, **GLOBAL_STATUS**, **SESSION_STATUS**, **GLOBAL_VARIABLES**, and **SESSION_VARIABLES**, that correspond to the output from the **SHOW {GLOBAL|SESSION} STATUS** and **SHOW {GLOBAL|SESSION} VARIABLES** statements.
- The **BINARY** keyword now is forbidden as a data type attribute in stored routines (for example, **DECLARE v1 VARCHAR(25) BINARY**), because **DECLARE** does not support collations, and in this context **BINARY** specifies the binary collation of the variable's character set. ([Bug#20701](#))

- The source distribution has been updated so that the UDF example can be compiled under Windows with CMake. See [Section 28.3.4.5, “Compiling and Installing User-Defined Functions”](#). (Bug#19121)
- `LOAD DATA INFILE` no longer causes an implicit commit for all storage engines. It now causes an implicit commit only for tables using the `NDB` storage engine. (Bug#11151)
- The `LOAD DATA FROM MASTER` and `LOAD TABLE FROM MASTER` statements are deprecated. See [Section 12.6.2.2, “LOAD DATA FROM MASTER Syntax”](#), for recommended alternatives. (Bug#18822, Bug#9125, Bug#12187, Bug#14399, Bug#15025, Bug#20596)
- `mysqldump` now has a `--flush-privileges` option. It causes `mysqldump` to emit a `FLUSH PRIVILEGES` statement after dumping the `mysql` database. This option should be used any time the dump contains the `mysql` database and any other database that depends on the data in the `mysql` database for proper restoration. (Bug#21424)
- The default value of the `tmp_table_size` system variable was lowered from 32MB to 16MB because it is bounded by the value of `max_heap_table_size`, which has a default of 16MB. (Bug#18875)
- The number of `InnoDB` threads is no longer limited to 1,000 on Windows. (Bug#22268)
- Memory consumption of the `InnoDB` data dictionary cache was roughly halved by cleaning up the data structures. (Bug#20877)
- Using `--with-debug` to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.
- A new system variable, `lc_time_names`, specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()` and `MONTHNAME()` functions. See [Section 5.10.9, “MySQL Server Locale Support”](#).

Bugs fixed:

- **Security fix:** On Linux, and possibly other platforms using case-sensitive filesystems, it was possible for a user granted rights on a database to create or access a database whose name differed only from that of the first by the case of one or more letters. (CVE-2006-4226, Bug#17647)
- **Security fix:** If a user has access to `MyISAM` table `t`, that user can create a `MERGE` table `m` that accesses `t`. However, if the user's privileges on `t` are subsequently revoked, the user can continue to access `t` by doing so through `m`. If this behavior is undesirable, you can start the server with the new `--skip-merge` option to disable the `MERGE` storage engine. (CVE-2006-4031, Bug#15195)
- **Security fix:** A stored routine created by one user and then made accessible to a different user using `GRANT EXECUTE` could be executed by that user with the privileges of the routine's definer. (CVE-2006-4227, Bug#18630)
- **Incompatible change:** For `utf8` columns, the full-text parser incorrectly considered several non-word punctuation and whitespace characters as word characters, causing some searches to return incorrect results. (Bug#19580)

The fix involves a change to the full-text parser, so any tables that have `FULLTEXT` indexes on `utf8` columns must be repaired with `REPAIR TABLE`:

```
REPAIR TABLE tbl_name QUICK;
```

- The `mysql_list_fields()` C API function returned the incorrect table name for views. (Bug#19671)
- An invalid `GRANT` statement for which `Ok` was returned on a replication master caused an error on the slave and replication to fail. (Bug#6774)
- `InnoDB` locking was improved by removing a gap lock for the case that you try to delete the same row twice within a transaction. (Bug#13544)
- Deleting entries from a large `MyISAM` index could cause index corruption when it needed to shrink. Deletes from an index can happen when a record is deleted, when a key changes and must be moved, and when a key must be un-inserted because of a duplicate key. This can also happen in `REPAIR TABLE` when a duplicate key is found and in `myisamchk` when sorting the records by an index. (Bug#22384)

- Setting `myisam_repair_threads` caused any repair operation on a `MyISAM` table to fail to update the cardinality of indexes, instead making them always equal to 1. (Bug#18874)
- `ALTER EVENT` statements including only a `COMMENT` clause failed with a syntax error on two platforms: Linux for S/390, and OS X 10.4 for 64-bit PPC. (Bug#23423)
- For row-based replication, log rotation could occur at an improper time. (Bug#21474)
- For row-based replication, the `BINLOG` command did not lock tables properly, causing a crash for some table types. (Bug#19459)
- The parser rejected queries that selected from a table twice using a `UNION` within a subquery. The parser now supports arbitrary subquery, join, and parenthesis operations within `EXISTS` subqueries. A limitation still exists for scalar subqueries: If the subquery contains `UNION`, the first `SELECT` of the `UNION` cannot be within parentheses. For example, `SELECT (SELECT a FROM t1 UNION SELECT b FROM t2)` will work, but `SELECT ((SELECT a FROM t1) UNION (SELECT b FROM t2))` will not. (Bug#14654)
- Incorrect type aggregation for `IN` and `CASE` expressions could lead to an incorrect result. (Bug#18360)
- When using row based replication, a `CREATE TABLE...SELECT` statement would be replicated, even if the table creation failed on the master (for example, due to a duplicate key failure). (Bug#20265)
- The optimizer did not take advantage of indexes on columns used for the second or third arguments of `BETWEEN`. (Bug#18165)
- Subqueries with aggregate functions but no `FROM` clause could return incorrect results. (Bug#21540)
- The `CSV` storage engine failed to detect some table corruption. (Bug#22080)
- For multiple-table `UPDATE` statements, storage engines were not notified of duplicate-key errors. (Bug#21381)
- Successive invocations of a `COUNT(*)` query containing a join on two `MyISAM` tables and a `WHERE` clause of the form `WHERE (table1.column1 = table2.column2) OR table2.column2 IS NULL` yielded different results. (Bug#21019)
- It was possible to provide the `ExtractValue()` function with input containing “tags” that were not valid XML; for example, it was possible to use tag names beginning with a digit, which are disallowed by the W3C’s XML 1.0 specification. Such cases caused the function to return “junk” output rather than an error message signalling the user as to the true nature of the problem. (Bug#20854)
- The presence of a subquery in the `ON` clause of a join in a view definition prevented the `MERGE` algorithm from being used for the view in cases where it should be allowed. (Bug#21646)
- `BIT` columns were not replicated properly under row-based replication. (Bug#22550)
- Conversion of values inserted into a `BIT` column could affect adjacent columns. (Bug#22271)
- The URL into the online manual that is printed in the stack trace message by the server was out of date. (Bug#21449)
- Incorrect results could be obtained from re-execution of a parametrized prepared statement or a stored routine with a `SELECT` that uses `LEFT JOIN` with a second table having only one row. (Bug#21081)
- `PROCEDURE ANALYSE()` returned incorrect values of `M FLOAT(M, D)` and `DOUBLE(M, D)`. (Bug#20305)
- Join conditions using index prefixes on `utf8` columns of `InnoDB` tables incorrectly ignored rows where the length of the actual value was greater than the length of the index prefix. (Bug#19960)
- On an `INSERT` into an updatable but non-insertable view, an error message was issued stating that the view was not updatable. Now the message says the view is not insertable-into. (Bug#5505)
- `INSERT DELAYED` did not honor `SET INSERT_ID` or the `auto_increment_*` system variables. (Bug#20627, Bug#20830)
- For character sets having a `mbmaxlen` value of 2, any `ALTER TABLE` statement changed `TEXT` columns to `MEDIUMTEXT`. (Bug#21620)
- A query that used `GROUP BY` and an `ALL` or `ANY` quantified subquery in a `HAVING` clause could trigger an assertion failure. (Bug#21853)
- For an `ENUM` column that used the `ucs2` character set, using `ALTER TABLE` to modify the column definition caused the default

value to be lost. (Bug#20108)

- An **UPDATE** that referred to a key column in the **WHERE** clause and activated a trigger that modified the column resulted in a loop. (Bug#20670)
- A loaded storage engine plugin did not load after a server restart. (Bug#21610)
- Creating a **TEMPORARY** table with the same name as an existing table that was locked by another client could result in a lock conflict for **DROP TEMPORARY TABLE** because the server unnecessarily tried to acquire a name lock. (Bug#21096)
- After **FLUSH TABLES WITH READ LOCK** followed by **UNLOCK TABLES**, attempts to drop or alter a stored routine failed with an error that the routine did not exist, and attempts to execute the routine failed with a lock conflict error. (Bug#21414)
- **mysql_com.h** unnecessarily referred to the **ulong** type. (Bug#22227)
- Incorporated some portability fixes into the definition of **__attribute__** in **my_global.h**. (Bug#2717)
- Linking the **pthread** library to single-threaded MySQL libraries caused **dlopen()** to fail at runtime on HP-UX. (Bug#18267)
- Loading a plugin caused any an existing plugin with the same name to be lost. (Bug#20615)
- In the package of pre-built time zone tables that is available for download at <http://dev.mysql.com/downloads/timezones.html>, the tables now explicitly use the **utf8** character set so that they work the same way regardless of the system character set value. (Bug#21208)
- The build process incorrectly tried to overwrite **sql/lex_hash.h**. This caused the build to fail when using a shadow link tree pointing to original sources that were owned by another account. (Bug#18888)
- **mysql_ftdump** produced bad counts for common words. (Bug#22326)
- The optimizer could make an incorrect index choice for indexes with a skewed key distribution. (Bug#22393)
- When records are merged from the insert buffer and the page needs to be reorganized, **InnoDB** used incorrect column length information when interpreting the records of the page. This caused a server crash due to apparent corruption of secondary indexes in **ROW_FORMAT=COMPACT** that contain prefix indexes of fixed-length columns. Data files should not be corrupted, but the crash was likely to repeat every time the server was restarted. (Bug#21638)
- Instance Manager didn't close the client socket file when starting a new **mysqld** instance. **mysqld** inherited the socket, causing clients connected to Instance Manager to hang. (Bug#12751)
- Using **GROUP_CONCAT()** on the result of a subquery in the **FROM** clause that itself used **GROUP_CONCAT()** could cause a server crash. (Bug#22015)
- Instance Manager had a race condition involving **mysqld** PID file removal. (Bug#22379)
- Execution of a prepared statement that uses an **IN** subquery with aggregate functions in the **HAVING** clause could cause a server crash. (Bug#22085)
- Selecting from a **MERGE** table could result in a server crash if the underlying tables had fewer indexes than the **MERGE** table itself. (Bug#21617, Bug#22937)
- A locking safety check in **InnoDB** reported a spurious error **stored_select_lock_type is 0 inside ::start_stmt()** for **INSERT ... SELECT** statements in **innodb_locks_unsafe_for_binlog** mode. The safety check was removed. (Bug#10746)
- **make install** tried to build files that should already have been built by **make all**, causing a failure if installation was performed using a different account than the one used for the initial build. (Bug#19738)
- The source distribution would not build on Windows due to a spurious dependency on **ib_config.h**. (Bug#22224)
- It was possible for a stored routine with a non-**latin1** name to cause a stack overrun. (Bug#21311)
- The server returns a more informative error message when it attempts to open a **MERGE** table that has been defined to use non-**MyISAM** tables. (Bug#10974)
- Within stored routines, some error messages were printed incorrectly. A non-null-terminated string was passed to a message-print-

ing routine that expected a null-terminated string. (Bug#20778)

- `SUBSTR()` results sometimes were stored improperly into a temporary table when multi-byte character sets were used. (Bug#20204)
- Certain malformed `INSERT` statements could crash the `mysql` client. (Bug#21142)
- `EXPLAIN EXTENDED` now shows a `filtered` column that is an estimated percentage of the examined rows that will be joined with the previous tables. This was added while dealing with a problem of MySQL choosing the wrong index for some queries. (Bug#14940)
- On Mac OS X, zero-byte `read()` or `write()` calls to an SMB-mounted filesystem could return a non-standard return value, leading to data corruption. Now such calls are avoided. (Bug#12620)
- With `TRADITIONAL` SQL mode, assignment of out-of-bound values and rounding of assigned values was done correctly, but assignment of the same numbers represented as strings sometimes was handled differently. (Bug#6147)
- The source distribution failed to compile when configured with the `--without-geometry` option. (Bug#12991)
- The source distribution failed to compile when configured with the `--with-libwrap` option. (Bug#18246)
- The feature of being able to recover a temporary table named `#sql_id` in InnoDB by creating a table named `rsql_id_recover_innodb_tmp_table` was broken by the introduction of the new identifier encoding in MySQL 5.1.6 (Bug#21313)
- `ALTER EVENT` in the body of a stored procedure led to a crash when the procedure was called. This affected only those `ALTER EVENT` statements which changed the interval of the event. (Bug#22397)
- A `DATE` can be represented as an integer (such as `20060101`) or as a string (such as `'2006.01.01'`). When a `DATE` (or `TIME`) column is compared in one `SELECT` against both representations, constant propagation by the optimizer led to comparison of `DATE` as a string against `DATE` as an integer. This could result in integer comparisons such as `2006` against `20060101`, erroneously producing a false result. (Bug#21475)
- When a statement used a stored function that inserted into an `AUTO_INCREMENT` column, the generated `AUTO_INCREMENT` value was not written into the binary log, so a different value could in some cases be inserted on the slave. (Bug#20341)
- A stored procedure that used `LAST_INSERT_ID()` did not replicate properly using statement-based binary logging. (Bug#20339)
- Use of the `--no-pager` option caused `mysql` to crash. (Bug#19363)
- For `INSERT ... ON DUPLICATE KEY UPDATE`, use of `VALUES(col_name)` within the `UPDATE` clause sometimes was handled incorrectly. (Bug#21555)
- When `event_scheduler` was set to `DISABLED`, its value was not displayed correctly by `SHOW VARIABLES` or `SELECT @@global.event_scheduler`. (Bug#22662)
- Row equalities (such as `WHERE (a,b) = (c,d)`) were not taken into account by the optimizer, resulting in slow query execution. Now they are treated as conjunctions of equalities between row elements. (Bug#16081)
- If the binary logging format was changed between the times when a locked table was modified and when it was unlocked, the binary log contents were incorrect. (Bug#20863)
- Column names supplied for a view created on a master server could be lost on a slave server. (Bug#19419)
- For a `MyISAM` table locked with `LOCK TABLES ...WRITE`, queries optimized using the `index_merge` method did not show rows inserted with the lock in place. (Bug#20256)
- Table aliases in multiple-table `DELETE` statements sometimes were not resolved. (Bug#21392)
- A query result could be sorted improperly when using `ORDER BY` for the second table in a join. (Bug#21302)
- The `--collation-server` server option was being ignored. With the fix for this problem, if you choose a non-default character set with `--character-set-server`, you should also use `--collation-server` to specify the collation. (Bug#15276)
- A function result in a comparison was replaced with a constant by the optimizer under some circumstances when this optimization was invalid. (Bug#21698)

- A subquery that uses an index for both the `WHERE` and `ORDER BY` clauses produced an empty result. (Bug#21180)
- If the `auto_increment_offset` setting causes MySQL to generate a value larger than the column's maximum possible value, the `INSERT` statement is accepted in strict SQL mode, whereas but should fail with an error. (Bug#20573)
- Queries containing a subquery that used aggregate functions could return incorrect results. (Bug#16792)
- Row-based replication failed when the query cache was enabled on the slave. (Bug#17620)
- The `index_merge/Intersection` optimizer could have a memory overrun when the number of table columns covered by an index is sufficiently large, possibly resulting in a server crash. (Bug#16201)
- The `MD5()`, `SHA1()`, and `ENCRYPT()` functions should return a binary string, but the result sometimes was converted to the character set of the argument. `MAKE_SET()` and `EXPORT_SET()` now use the correct character set for their default separators, resulting in consistent result strings which can be coerced according to normal character set rules. (Bug#20536)
- `EXPLAIN` sometimes returned an incorrect `select_type` for a `SELECT` from a view, compared to the `select_type` for the equivalent `SELECT` from the base table. (Bug#5500)
- An `InnoDB` mutex was not acquired and released under the same condition, leading to deadlock in some rare situations involving XA transactions. (Bug#21833)
- With row-based replication, replicating a statement to a slave where the table had additional columns relative to the master table did not work. (Bug#19069)
- For a `MyISAM` table with a `FULLTEXT` index, compression with `myisampack` or a check with `myisamchk` after compression resulted in table corruption. (Bug#19702)
- With `max_sp_recursion` set to 0, a stored procedure that executed a `SHOW CREATE PROCEDURE` statement for itself triggered a recursion limit exceeded error, though the statement involves no recursion. (Bug#21416)
- `mysqldump` did not add version-specific comments around `WITH PARSER` and `TABLESPACE ... STORAGE DISK` clauses for `CREATE TABLE` statements, causing the dump file to fail when loaded into older servers. (Bug#20841)
- `BIN()`, `OCT()`, and `CONV()` did not work with BIT values. (Bug#15583)
- The optimizer could produce an incorrect result after `AND` with collations such as `latin1_german2_ci`, `utf8_czech_ci`, and `utf8_lithianian_ci`. (Bug#9509)
- The server could crash for the second execution of a function containing a `SELECT` statement that uses an aggregating `IN` subquery. (Bug#21493)
- `UPGRADE` was treated as a reserved word, although it is not. (Bug#21772)
- Database and table names have a maximum length of 64 characters (even if they contain multi-byte characters), but were being truncated to 64 bytes. (Bug#21432) An additional fix was made in MySQL 5.1.18.
- Usernames have a maximum length of 16 characters (even if they contain multi-byte characters), but were being truncated to 16 bytes. (Bug#20393)
- A query could produce different results with and without an index, if the `WHERE` clause contained a range condition that used an invalid `DATETIME` constant. (Bug#16249)
- `COUNT(*)` queries with `ORDER BY` and `LIMIT` could return the wrong result. (Bug#21787)

Note

This problem was introduced by the fix for Bug#9676, which limited the rows stored in a temporary table to the `LIMIT` clause. This optimization is not applicable to non-group queries with aggregate functions. The current fix disables the optimization in such cases.

- Memory overruns could occur for certain kinds of subqueries. (Bug#21477)
- Adding `ORDER BY` to a `SELECT DISTINCT(expr)` query could produce incorrect results. (Bug#21456)
- Memory used by scheduled events was not freed when the events were dropped. (Bug#18683)

- A scheduled event that took longer to execute than the length of time scheduled between successive executions could “skip” executions. For example, an event defined with `EVERY 1 SECOND` — but which required longer than 1 second to complete — might be executed only once every 2 seconds. (Bug#16417)
- When used in the `DO` clause of a `CREATE EVENT` statement, the statements `CREATE EVENT`, `CREATE FUNCTION`, and `CREATE PROCEDURE` caused the server to crash. (These statements are not permitted inside `CREATE EVENT`.) (Bug#16409, Bug#18896)
- A subselect used in the `ON SCHEDULE` clause of a `CREATE EVENT` or `ALTER EVENT` statement caused the server to crash, rather than producing an error as expected. (Bug#16394)
- `mysql` displayed an empty string for `NULL` values. (Bug#21618)
- `mysql_upgrade` produced a malformed `upgrade_defaults` file by overwriting the `[client]` group header with a `password` option. This prevented `mysqlcheck` from running successfully when invoked by `mysql_upgrade`. (Bug#21011)
- `mysql_config --libmysqld-libs` did not produce any SSL options necessary for linking `libmysqld` with SSL support enabled. (Bug#21239)
- `yaSSL` had a conflicting definition for `socklen_t` on `hurd-i386` systems. (Bug#22326)
- On Windows, inserting into a `MERGE` table after renaming an underlying `MyISAM` table caused a server crash. (Bug#20789)
- `character_set_results` can be `NULL` to signify “no conversion,” but some code did not check for `NULL`, resulting in a server crash. (Bug#21913)
- If a partitioned `InnoDB` table contained an `AUTO_INCREMENT` column, a `SHOW` statement could cause an assertion failure with more than one connection. (Bug#20493)
- Running `InnoDB` with many concurrent threads could cause memory corruption and a seg fault due to a bug introduced in MySQL 5.1.11. (Bug#20213)
- Using `DROP TABLE` with concurrent queries causes `mysqld` to crash. (Bug#21784)
- The `ExtractValue()` function did not accept XML tag names containing a period (`.`) character. (Bug#20795)
- For table-format output, `mysql` did not always calculate columns widths correctly for columns containing multi-byte characters in the column name or contents. (Bug#17939)
- Identifiers with embedded escape characters were not handled correctly by some `SHOW` statements due to some old code that was doing some extra unescaping. (Bug#19874)
- `InnoDB` was slow with more than 100,000 `.idb` files. (Bug#21112)
- `SHOW INNODB STATUS` contained some duplicate output. (Bug#21113)
- After an `INSERT ... ON DUPLICATE KEY UPDATE` statement that updated an existing row, `LAST_INSERT_ID()` could return a value not in the table. (Bug#11460)
- Selecting from `INFORMATION_SCHEMA.FILES` could crash the server. (Bug#21676)
- Using cursors with `READ COMMITTED` isolation level could cause `InnoDB` to crash. (Bug#19834)
- A server or network failure with an open client connection would cause the client to hang even though the server was no longer available. (Bug#9678)
- Using `ALTER TABLE ... REORGANIZE PARTITIONS` to reduce the number of subpartitions to 1 caused the server to crash. (Bug#21210)
- **NDB Cluster**: The `ndb_size.pl` script did not account for `TEXT` and `BLOB` column values correctly. (Bug#21204)
- **NDB Cluster**: Data nodes added while the cluster was running in single user mode were all assigned node ID 0, which could later cause multiple node failures. Adding of nodes in single user mode is no longer possible. (Bug#20395)
- **NDB Cluster**: Attempting to create an `NDB` table on a MySQL with an existing non-Cluster table with the same name in the same database could result in data loss or corruption. Now, if such a table is encountered during autodiscovery, a warning is written to the

error log of the affected `mysqld`, and the local table is overwritten. ([Bug#21378](#))

- **NDB Cluster** (NDB API): Inactivity timeouts for scans were not correctly handled. ([Bug#23107](#))
- **NDB Cluster** (NDB API): Attempting to read a nonexistent tuple using `Commit` mode for `NdbTransaction::execute()` caused node failures. ([Bug#22672](#))
- **NDB Cluster** (NDB API): The inclusion of `my_config.h` in `NdbApi.h` required anyone wishing to write NDB API applications against MySQL 5.1 to have a complete copy of the 5.1 sources. ([Bug#21253](#))
- **NDB Cluster** (NDB API): The `NdbOperation::getBlobHandle()` method, when called with the name of a nonexistent column, caused a segmentation fault. ([Bug#21036](#))
- **NDB Cluster**: The `--help` output from NDB binaries did not include file-related options. ([Bug#21994](#))
- **NDB Cluster**: The node recovery algorithm was missing a version check for tables in the `ALTER_TABLE_COMMITTED` state (as opposed to the `TABLE_ADD_COMMITTED` state, which has the version check). This could cause inconsistent schemas across nodes following node recovery. ([Bug#21756](#))
- **NDB Cluster**: The output for the `--help` option used with NDB executable programs (`ndbd`, `ndb_mgm`, `ndb_restore`, `ndb_config`, and so on) referred to the `Ndb.cfg` file, instead of `my.cnf`. ([Bug#21585](#))
- **NDB Cluster**: Partition distribution keys were updated only for the primary and starting replicas during node recovery. This could lead to node failure recovery for clusters having an odd number of replicas. ([Bug#21535](#))

Note

We recommend values for `NumberOfReplicas` that are even powers of 2, for best results.

- **NDB Cluster**: The `ndb_mgm` management client did not set the exit status on errors, always returning 0 instead. ([Bug#21530](#))
- **NDB Cluster**: Cluster logs were not rotated following the first rotation cycle. ([Bug#21345](#))
- **NDB Cluster**: Condition pushdown did not work correctly with `DATETIME` columns. ([Bug#21056](#))
- **NDB Cluster**: Under some circumstances, local checkpointing would hang, keeping any unstarted nodes from being started. ([Bug#20895](#))
- **NDB Cluster**: Using an invalid node ID with the management client `STOP` command could cause `ndb_mgm` to hang. ([Bug#20575](#))
- **NDB Cluster**: In some cases where `SELECT COUNT(*)` from an NDB table should have yielded an error, `MAX_INT` was returned instead. ([Bug#19914](#))
- **NDB Cluster**: Following the restart of an MGM node, the Cluster management client did not automatically reconnect. ([Bug#19873](#))
- **NDB Cluster**: Error messages given when trying to make online changes parameters such as `NoOfReplicas` that can only be changed via a complete shutdown and restart of the cluster did not indicate the true nature of the problem. ([Bug#19787](#))
- **NDB Cluster**: `ndb_restore` did not always make clear that it had recovered successfully from temporary errors while restoring a cluster backup. ([Bug#19651](#))
- **NDB Cluster**: In rare situations with resource shortages, a crash could result from insufficient `IndexScanOperations`. ([Bug#19198](#))
- **NDB Cluster**: `ndb_mgm -e show | head` would hang after displaying the first 10 lines of output. ([Bug#19047](#))
- **NDB Cluster**: The error returned by the cluster when too many nodes were defined did not make clear the nature of the problem. ([Bug#19045](#))
- **NDB Cluster**: A problem with takeover during a system restart caused ordered indexes to be rebuilt incorrectly. This also adversely affected Cluster Replication. ([Bug#15303](#))
- **NDB Cluster**: Some queries involving joins on very large NDB tables could crash the MySQL server. ([Bug#21059](#))

- **NDB Cluster** (Disk Data): `mysqldump` did not back up tablespace or log file group information for Disk Data tables correctly. (Specifically, `UNDO_BUFFER_SIZE` and `INITIAL_SIZE` values were misreported.) Trying to restore from such a backup would produce error 1296 (`GOT ERROR 1504 'OUT OF LOGBUFFER MEMORY' FROM NDB`). (Bug#20809)
- **NDB Cluster** (Disk Data): The `INFORMATION_SCHEMA.FILES` table showed incorrect values in the `EXTENT_SIZE`, `FREE_EXTENTS`, and `TOTAL_EXTENTS` columns for UNDO log files. (Bug#20073)
- **NDB Cluster** (Disk Data): Deletes from Disk Data tables used a non-optimal scan to find the rows to be deleted, resulting in poor performance. The fix causes disk order rather than memory order to be used, and can improve performance of Disk Data deletes by up to ~300% in some cases. (Bug#17929)
- **NDB Cluster**: A scan timeout returned Error 4028 (`NODE FAILURE CAUSED ABORT OF TRANSACTION`) instead of Error 4008 (`NODE FAILURE CAUSED ABORT OF TRANSACTION...`). (Bug#21799)
- **NDB Cluster**: The message `ERROR 0 IN READAUTOINCREMENTVALUE(): NO ERROR` was written to the error log whenever `SHOW TABLE STATUS` was performed on a Cluster table that did not have an `AUTO_INCREMENT` column. (Bug#21033)
- **NDB Cluster** (Direct APIs): The `storage/ndb` directory was missing from the server binary distribution, making it impossible to compile NDB API and MGM API applications. This directory can be found as `/usr/include/storage/ndb` after installing that distribution. (Bug#21955)
- **NDB Cluster**: `ndb_size.pl` and `ndb_error_reporter` were missing from RPM packages. (Bug#20426)
- The `ndb_mgm` program was included in both the `MySQL-ndb-tools` and `MySQL-ndb-management` RPM packages, resulting in a conflict if both were installed. Now `ndb_mgm` is included only in `MySQL-ndb-tools`. (Bug#21058)
- `libmysqld` produced some warnings to `stderr` which could not be silenced. These warnings now are suppressed. (Bug#13717)
- If a column definition contained a character set declaration, but a `DEFAULT` value began with an introducer, the introducer character set was used as the column character set. (Bug#20695)
- If a query had a condition of the form `tableX.key = tableY.key`, which participated in equality propagation and also was used for `ref` access, then early `ref`-access `NULL` filtering was not performed for the condition. This could make query execution slower. (Bug#19649)
- The optimizer sometimes produced an incorrect row-count estimate after elimination of `CONST` tables. This resulted in choosing extremely inefficient execution plans in some cases when distribution of data in joins were skewed. (Bug#21390)
- Query results could be incorrect if the `WHERE` clause contained `t.key_part NOT IN (val_list)`, where `val_list` is a list of more than 1000 constants. (Bug#21282)
- `STR_TO_DATE()` sometimes would return `NULL` if the `%D` format specifier was not the last specifier in the format string. (Bug#20987)
- On Windows, a definition for `mysql_set_server_option()` was missing from the C client library. (Bug#16513)
- The `myisam_stats_method` variable was mishandled when set from an option file or on the command line. (Bug#21054)
- The optimizer assumed that if `(a=x AND b=x)` is true, `(a=x AND b=x) AND a=b` is also true. But that is not always so if `a` and `b` have different data types. (Bug#21159)
- **InnoDB** did not honor `IGNORE INDEX`, which prevented using `IGNORE INDEX` in cases where an index sort would be slower than a filesort. (Bug#21174)
- For connections that required a `SUBJECT` value, a check was performed to verify that the value was correct, but the connection was not refused if not. (Bug#20411)
- Some Linux-x86_64-icc packages (of previous releases) mistakenly contained 32-bit binaries. Only `ICC` builds are affected, not `gcc` builds. Solaris and FreeBSD x86_64 builds are not affected. (Bug#22238)
- `INSERT ... SELECT` sometimes generated a spurious `Column count doesn't match value count` error. (Bug#21774)
- Some user-level errors were being written to the server's error log, which is for server errors. (Bug#20402)
- to the client could have an empty column name. When using tables created under MySQL 4.1 with a 5.0 server, if the tables con-

tained `VARCHAR` columns, for some queries the metadata sent to the client could have an empty column name. (Bug#14897)

- On 64-bit systems, use of the `cp1250` character set with a primary key column in a `LIKE` clause caused a server crash for patterns having letters in the range 128..255. (Bug#19741)
- `ORDER BY RAND() LIMIT 1` always set a user variable to the last possible value from the table. (Bug#16861)
- `N'xxx'` and `_utf8'xxx'` were not treated as equivalent because `N'xxx'` failed to unescape backslashes (`\`) and doubled apostrophe/single quote characters (`' '`). (Bug#17313)
- A subquery in the `WHERE` clause of the outer query and using `IN` and `GROUP BY` returned an incorrect result. (Bug#16255)
- When `NOW()` was used in a `BETWEEN` clause of the definition for a view, it was replaced with a constant in the view. (Bug#15950)
- A stored procedure with a `CONTINUE` handler that encountered an error continued to execute a statement that caused an error, rather with the next statement following the one that caused the error. (Bug#8153)
- `libmysqlclient` defined a symbol `BN_bin2bn` which belongs to OpenSSL. This could break applications that also linked against OpenSSL's `libcrypto` library. The fix required correcting an error in a build script that was failing to add rename macros for some functions. (Bug#21930)
- Queries that used the `index_merge` and `sort_union` methods to access an `InnoDB` table could produce inaccurate results. This issue was introduced in MySQL 5.1.10 when a new handler and bitmap interface was implemented. (Bug#21277)
- The `SELECT` privilege was required for an insert on a view, instead of the `INSERT` privilege. (Bug#21261)

Note

This fixes a regression that was introduced by the fix for Bug#20989.

- Running `SHOW MASTER LOGS` at the same time as binary log files were being switched would cause `mysqld` to hang. (Bug#21965)
- Building `mysql` on Windows with CMake 2.4 would fail to create `libmysqld` correctly. (Bug#20907)
- The server's handling of the number of partitions or subpartitions specified in a `PARTITIONS` or `SUBPARTITIONS` clause was changed. Beginning with this release, the number of partitions must:
 - be a positive, non-zero integer
 - not have any leading zeroes
 - not be an expression

Also beginning with this version, no attempt is made to convert, truncate, or evaluate a `PARTITIONS` or `SUBPARTITIONS` value; instead, the `CREATE TABLE` or `ALTER TABLE` statement containing the `PARTITIONS` or `SUBPARTITIONS` clause now fails with an appropriate error message. (Bug#15890)

- A misleading error message was displayed when attempting to define a unique key that was not valid for a partitioned table. (Bug#21862)
- Errors could be generated during the execution of certain prepared statements that ran queries on partitioned tables. (Bug#21658)
- Using relative paths for `DATA DIRECTORY` or `INDEX DIRECTORY` with a partitioned table generated a warning rather than an error, and caused “junk” files to be created in the server's data directory. (Bug#21350)
- Using `EXPLAIN PARTITIONS` with a query on a table whose partitioning expression was based on the value of a `DATE` column could sometimes cause the server to crash. (Bug#21339)
- Running `SHOW TABLE STATUS` on any `InnoDB` table having at least one record could crash the server. Note that this was not due to any issue in the `InnoDB` storage engine, but rather with `AUTO_INCREMENT` handling in the partitioning code — however, the table did not have to have an `AUTO_INCREMENT` column for the bug to manifest. (Bug#21173)
- Some `ALTER TABLE` statements affecting a table's subpartitioning could hang. (Bug#21143)
- Scheduled events that invoked stored procedures executing DDL operations on partitioned tables could crash the server.

[Bug#20548](#))

- The yaSSL library bundled with `libmysqlclient` had some conflicts with OpenSSL. Now macros are used to rename the conflicting symbols to have a prefix of `ya`. ([Bug#19810](#))
- It is possible to create `MERGE` tables into which data cannot be inserted (by not specifying a `UNION` clause. However, when an insert was attempted, the error message was confusing. Now an error occurs indicating that the table is read-only. ([Bug#17766](#))
- User-created tables having a name beginning with `#sql` were not visible to `SHOW TABLES` and could collide with internal temporary table names. Now they are not hidden and do not collide. ([Bug#1405](#))
- A `NUL` byte within a prepared statement string caused the rest of the string not to be written to the query log, allowing logging to be bypassed. ([Bug#21813](#))
- `mysql_upgrade` created temporary files in a possibly insecure way. ([Bug#21224](#))
- Some prepared statements caused a server crash when executed a second time. ([Bug#21166](#))
- With `query_cache_type` set to 0, `RESET QUERY CACHE` was very slow and other threads were blocked during the operation. Now a cache reset is faster and non-blocking. ([Bug#21051](#))
- When `DROP DATABASE` or `SHOW OPEN TABLES` was issued while concurrently issuing `DROP TABLE` (or `RENAME TABLE`, `CREATE TABLE LIKE` or any other statement that required a name lock) in another connection, the server crashed. ([Bug#19403](#), [Bug#21216](#))
- Use of zero-length variable names caused a server crash. ([Bug#20908](#))
- Prepared statements caused general log and server memory corruption. ([Bug#14346](#))
- `mysqldump` incorrectly tried to use `LOCK TABLES` for tables in the `INFORMATION_SCHEMA` database. ([Bug#21527](#))
- Use of the `--prompt` option or `prompt` command caused `mysql` to be unable to connect to the Instance Manager. ([Bug#17485](#))
- The server crashed if it tried to access a `CSV` table for which the data file had been removed. ([Bug#15205](#))
- `CREATE USER` did not respect the 16-character username limit. ([Bug#10668](#))
- Creating a partitioned table that used the `InnoDB` storage engine and then restarting `mysqld` with `--skip-innodb` caused MySQL to crash. ([Bug#20871](#))
- In mixed-format binary logging mode, stored functions, triggers, and views that use functions in their body that require row-based logging did not replicate reliably because the logging did not switch from statement-based to row-based format. For example, `INSERT INTO t SELECT FROM v`, where `v` is a view that selects `UUID()` could cause problems. This limitation has been removed. ([Bug#20930](#))
- For user-defined functions created with `CREATE FUNCTION`, the `DEFINER` clause is not legal, but no error was generated. ([Bug#21269](#))
- `mysqld --flush` failed to flush `MyISAM` table changes to disk following an `UPDATE` statement for which no updated column had an index. ([Bug#20060](#))
- In mixed-format binary logging mode, stored functions, triggers, and views that use functions in their body that require row-based logging did not replicate reliably because the logging did not switch from statement-based to row-based format. For example, `INSERT INTO t SELECT FROM v`, where `v` is a view that selects `UUID()` could cause problems. This limitation has been removed. ([Bug#20930](#))
- When not running in strict mode, the server failed to convert the invalid years portion of a `DATE` or `DATETIME` value to `'0000'` when inserting it into a table. ([Bug#19370](#))

This patch was reverted in MySQL 5.1.18.

- The dropping of a temporary table whose name contained a backtick (```) character was not correctly written to the binary log, which also caused it not to be replicated correctly. ([Bug#19188](#))
- Intermediate tables created during the execution of an `ALTER TABLE` statement were visible in the output of `SHOW TABLES`. ([Bug#18775](#))

- When setting a column to its implicit default value as the result of inserting a `NULL` into a `NOT NULL` column as part of a multi-row insert or `LOAD DATA` operation, the server returned a misleading warning message. (Bug#14770)
- The `--with-collation` option was not honored for client connections. (Bug#7192)
- Users who had the `SHOW VIEW` privilege for a view and privileges on one of the view's base table could not see records in `INFORMATION_SCHEMA` tables relating to the base table. (Bug#20543)
- An issue with yaSSL prevented Connector/J clients from connecting to the server using a certificate. (Bug#19705)
- Some server errors were not reported to the client, causing both to try to read from the connection until a hang or crash resulted. (Bug#16581)
- `FEDERATED` tables raised invalid duplicate key errors when attempting on one server to insert rows having the same primary key values as rows that had been deleted from the linked table on the other server. (Bug#18764)
- The C API failed to return a status message when invoking a stored procedure. (Bug#15752)
- `AUTHORS` and `CONTRIBUTORS` were not treated as reserved words. (Bug#19939)
- Stored procedures did not use the character set defined for the database in which they were created. (Bug#16676)
- `CREATE PROCEDURE`, `CREATE FUNTION`, `CREATE TRIGGER`, and `CREATE VIEW` statements containing multi-line comments (`/* ... */`) could not be replicated. (Bug#20438)
- The final parenthesis of a `CREATE INDEX` statement occurring in a stored procedure was omitted from the binary log when the stored procedure was called. (Bug#19207)
- Attempting to insert a string of greater than 4096 bytes into a `FEDERATED` table resulted in the error `ERROR 1296 (HY000) AT LINE 2: GOT ERROR 10000 'ERROR ON REMOTE SYSTEM: 1054: UNKNOWN COLUMN 'STRING-VALUE' FROM FEDERATED.` This error was raised regardless of the type of column involved (`VARCHAR`, `TEXT`, and so on.) (Bug#17608)
- Performance during an import on a table with a trigger that called a stored procedure was severely degraded. (Bug#21013)
- Repeated `DROP TABLE` statements in a stored procedure could sometimes cause the server to crash. (Bug#19399)
- On 64-bit Windows, a missing table generated error 1017, not the correct value of 1146. (Bug#21396)
- The same trigger error message was produced under two conditions: The trigger duplicated an existing trigger name, or the trigger duplicated an existing combination of action and event. Now different messages are produced for the two conditions so as to be more informative. (Bug#10946)
- The value returned by a stored function returning a string value was not of the declared character set. (Bug#16211)
- `FLUSH TABLES` followed by a `LOCK TABLES` statement to lock a log table and a non-log table caused an infinite loop and high CPU use. Now `FLUSH TABLES` ignores log tables. To flush the log tables, use `FLUSH LOGS` instead. (Bug#20139)
- If a filename was specified for the `--log` or `--log-slow_queries` options but the server was logging to tables and not files, the server produced no error message. (Bug#17599)
- `mysqlcheck` tried to check views instead of ignoring them. (Bug#16502)
- Long multiple-row `INSERT` statements could take a very long time for some multi-byte character sets. (Bug#15811)
- For `mysql`, escaping with backslash sometimes did not work. (Bug#20103)
- Under certain circumstances, `AVG(key_val)` returned a value but `MAX(key_val)` returned an empty set due to incorrect application of `MIN()/MAX()` optimization. (Bug#20954)
- Using aggregate functions in subqueries yielded incorrect results under certain circumstances due to incorrect application of `MIN()/MAX()` optimization. (Bug#20792)
- A query using `WHERE column = constant OR column IS NULL` did not return consistent results on successive invocations. The `column` in each part of the `WHERE` clause could be either the same column, or two different columns, for the effect to be observed. (Bug#21019)

- The `PASSWORD()` function returned invalid results when used in some `UNION` queries. (Bug#16881)
- `USE` did not refresh database privileges when employed to re-select the current database. (Bug#10979)
- A query using `WHERE NOT (column < ANY (subquery))` yielded a different result from the same query using the same `column` and `subquery` with `WHERE (column > ANY (subquery))`. (Bug#20975)
- A user variable set to a value selected from an unsigned column was stored as a signed value. (Bug#7498)
- `SELECT` statements using `GROUP BY` against a view could have missing columns in the output when there was a trigger defined on one of the base tables for the view. (Bug#20466)
- A `SELECT` with a subquery that was bound to the outer query over multiple columns returned different results when a constant was used instead of one of the dependant columns. (Bug#18925)
- `InnoDB`: Quoted Unicode identifiers were not handled correctly. This included names of tables, columns, and foreign keys. (Bug#18800)
- A stored procedure that created and invoked a prepared statement was not executed when called in a `mysqld` init-file. (Bug#17843)
- Using the extended syntax for `TRIM()` — that is, `TRIM(... FROM ...)` — in a `SELECT` statement defining a view caused an invalid syntax error when selecting from the view. (Bug#17526)
- Assignments of values to variables of type `TEXT` were handled incorrectly in stored routines. (Bug#17225)
- When performing a `GROUP_CONCAT()`, the server transformed `BLOB` columns `VARCHAR` columns, which could cause erroneous results when using Connector/J and possibly other MySQL APIs. (Bug#16712)
- The type of the value returned by the `VARIANCE()` function varied according to the type of the input value. The function should always return a `DOUBLE` value. (Bug#10966)
- Performing an `INSERT` on a view that was defined using a `SELECT` that specified a collation and a column alias caused the server to crash (Bug#21086).
- A query of the form shown here caused the server to crash:

```
SELECT * FROM t1 NATURAL JOIN (
  t2 JOIN (
    t3 NATURAL JOIN t4,
    t5 NATURAL JOIN t6
  )
  ON (t3.id3 = t2.id3 AND t5.id5 = t2.id5)
);
```

(Bug#21007)

- `myisam_ftdump` would fail when trying to open a MyISAM index file that you did not have write permissions to access, even though the command would only be reading from the file. (Bug#17122)
- `REPLACE ... SELECT` for a view required the `INSERT` privilege for tables other than the table being modified. (Bug#20989)
- `mysqldump` sometimes did not select the correct database before trying to dump views from it, resulting in an empty result set that caused `mysqldump` to die with a segmentation fault. (Bug#21014)
- With mixed-format binary logging, `INSERT DELAYED` statements were logged using statement-based logging, and they did not replicate properly for statements that used values such as `UUID()`, `RAND()`, or user-defined variables that require row-based logging. To correct this, the `DELAYED` handler thread now switches to row-based logging if the logging format is mixed. (Bug#20633, Bug#20649)
- Using `EXPLAIN PARTITIONS` with a `UNION` query could crash the server. This could occur whether or not the query actually used any partitioned tables. (Bug#20484)
- Partition pruning could cause incorrect results from queries, such missing rows, when the partitioning expression relied on a `BIGINT UNSIGNED` column. (Bug#20257)
- The implementation for `UNCOMPRESS()` did not indicate that it could return `NULL`, causing the optimizer to do the wrong thing. (Bug#18539)

- `TIMESTAMPDIFF()` examined only the date and ignored the time when the requested difference unit was months or quarters. (Bug#16226)
- `perror` did not properly report NDB error codes. (Bug#16561)
- `mysqlimport` sends a `set @@character_set_database=binary` statement to the server, but this is not understood by pre-4.1 servers. Now `mysqlimport` encloses the statement within a `/*!40101 ... */` comment so that old servers will ignore it. (Bug#15690)
- The character set was not being properly initialized for `CAST()` with a type like `CHAR(2) BINARY`, which resulted in incorrect results or even a server crash. (Bug#17903)
- For ODBC compatibility, MySQL supports use of `WHERE col_name IS NULL` for `DATE` or `DATETIME` columns that are `NOT NULL`, to allow column values of `'0000-00-00'` or `'0000-00-00 00:00:00'` to be selected. However, this was not working for `WHERE` clauses in `DELETE` statements. (Bug#8143)
- The `--master-data` option for `mysqldump` requires certain privileges, but `mysqldump` generated a truncated dump file without producing an appropriate error message or exit status if the invoking user did not have those privileges. (Bug#21215)
- `ALTER VIEW` did not retain existing values of attributes that had been originally specified but were not changed in the `ALTER VIEW` statement. (Bug#21080)
- `mysql` crashed for very long arguments to the `connect` command. (Bug#21042)
- `perror` crashed on Solaris due to `NULL` return value of `strerror()` system call. (Bug#20145)
- The `query` command for `mysqltest` did not work. (Bug#19890)
- For certain queries, the server incorrectly resolved a reference to an aggregate function and crashed. (Bug#20868)
- When executing a `SELECT` with `ORDER BY` on a view that is constructed from a `SELECT` statement containing a stored function, the stored function was evaluated too many times. (Bug#19862)
- A `SELECT` that used a subquery in the `FROM` clause that did not select from a table failed when the subquery was used in a join. (Bug#21002)
- Subqueries on `INFORMATION_SCHEMA` tables could erroneously return an empty result. (Bug#21231)
- Issuing a `SHOW CREATE FUNCTION` or `SHOW CREATE PROCEDURE` statement without sufficient privileges could crash the `mysql` client. (Bug#20664)
- In a view defined with `SQL SECURITY DEFINER`, the `CURRENT_USER()` function returned the invoker, not the definer. (Bug#20570)
- `DATE_ADD()` and `DATE_SUB()` returned `NULL` when the result date was on the day `'9999-12-31'`. (Bug#12356)
- For a `DATE` parameter sent via a `MYSQL_TIME` data structure, `mysql_stmt_execute()` zeroed the hour, minute, and second members of the structure rather than treating them as read-only. (Bug#20152)
- The `DATA DIRECTORY` table option did not work for `TEMPORARY` tables. (Bug#8706)
- If the files for an open table were removed at the OS level (external to the server), the server exited with an assertion failure. (Bug#16532)
- Some memory leaks in the `libmysqld` embedded server were corrected. (Bug#16017)
- With the `auto_increment_increment` system variable set larger than 1, if the next generated `AUTO_INCREMENT` value would be larger than the column's maximum value, the value would be clipped down to that maximum value and inserted, even if the resulting value would not be in the generated sequence. This could cause problems for master-master replication. Now the server clips the value down to the previous value in the sequence, which correctly produces a duplicate-key error if that value already exists in the column. (Bug#20524)
- If a table on a slave server had a higher `AUTO_INCREMENT` counter than the corresponding master table (even though all rows of the two tables were identical), in some cases `REPLACE` or `INSERT ... ON DUPLICATE KEY UPDATE` would not replicate properly using statement-based logging. (Different values would be inserted on the master and slave.) (Bug#20188)

- `mysqlslap` did not enable the `CLIENT_MULTI_RESULTS` flag when connecting, which is necessary for executing stored procedures. (Bug#20365)
- When creating a table using `CREATE ... SELECT` and a stored procedure, there would be a mismatch between the binary log and transaction cache which would cause a server crash. (Bug#21039)
- When run with the `--use-threads` option, `mysqlimport` returned a random exit code. (Bug#21188)
- The effect of a stored function or trigger that caused `AUTO_INCREMENT` values to be generated for multiple tables was not logged properly if statement-based logging was used. Only the first table's value was logged, causing replication to fail. Under mixed logging format, this is dealt with by switching to row-based logging for the function or trigger. For statement-based logging, this remains a problem. (Bug#19630)
- Changing the definition of a `DECIMAL` column with `ALTER TABLE` caused loss of column values. (Bug#18014)
- Under heavy load (executing more than 1024 simultaneous complex queries), a problem in the code that handles internal temporary tables could lead to writing beyond allocated space and memory corruption. Use of more than 1024 simultaneous cursors server wide also could lead to memory corruption. (This applies both to stored procedure and C API cursors.) (Bug#21206)
- A race condition during slave server shutdown caused an assert failure. (Bug#20850)
- `mysqldump` produced a malformed dump file when dumping multiple databases that contained views. (Bug#20221)
- Partitions were represented internally as the wrong data type, which led in some cases to failures of queries such as `SELECT COUNT(*) FROM INFORMATION_SCHEMA.PARTITIONS WHERE PARTITION_NAME = 'partition_name'`. (Bug#20340)
- Searches against a `ZEROFILL` column of a partitioned table could fail when the `ZEROFILL` column was part of the table's partitioning key. (Bug#20733)
- In mixed binary logging mode, a temporary switch from statement-based logging to row-based logging occurs when storing a row that uses a function such as `UUID()` into a temporary table. However, temporary table changes are not written to the binary log under row-based logging, so the row does not exist on the slave. A subsequent select from the temporary table to a non-temporary table using statement-based logging works correctly on the master, but not on the slave where the row does not exist. The fix for this is that replication does not switch back from row-based logging to statement-based logging until there are no temporary tables for the session. (Bug#20499)
- Re-executing a stored procedure with a complex stored procedure cursor query could lead to a server crash. (Bug#15217)
- Views created from prepared statements inside of stored procedures were created with a definition that included both `SQL_CACHE` and `SQL_NO_CACHE`. (Bug#17203)
- Updating a column of a `FEDERATED` table to `NULL` sometimes failed. (Bug#16494)
- Performing `INSERT ... SELECT ... JOIN ... USING` without qualifying the column names caused `ERROR 1052 "column 'x' in field list is ambiguous"` even in cases where the column references were unambiguous. (Bug#18080)
- Closing of temporary tables failed if binary logging was not enabled. (Bug#20919)
- For statements that have a `DEFINER` clause such as `CREATE TRIGGER` or `CREATE VIEW`, long usernames or hostnames could cause a buffer overflow. (Bug#16899)
- `mysqldump` would not dump views that had become invalid because a table named in the view definition had been dropped. Instead, it quit with an error message. Now you can specify the `--force` option to cause `mysqldump` to keep going and write a SQL comment containing the view definition to the dump output. (Bug#17371)
- **InnoDB** (Partitioning): Updating an **InnoDB** table using `HASH` partitioning with a composite primary key would cause the server to hang. (Bug#20852)
- Old partition and subpartition files were not always removed following `ALTER TABLE ... REORGANIZE PARTITION` statements. (Bug#20770)
- Merging multiple partitions having subpartitions into a single partition with subpartitions, or splitting a single partition having subpartitions into multiple partitions with subpartitions, could sometimes crash the server. These issues were associated with a failure reported in the `partition_range` test. (Bug#20766, Bug#20767, Bug#20893, Bug#21357)

- Some queries using `ORDER BY ... DESC` on subpartitioned tables could crash the server. (Bug#20389)
- Referring to a stored function qualified with the name of one database and tables in another database caused a “table doesn't exist” error. (Bug#18444)
- For `NDB` and possibly `InnoDB` tables, a `BEFORE UPDATE` trigger could insert incorrect values. (Bug#18437)
- For multiple `INSERT DELAYED` statements executed in a batch by the delayed-insert handler thread, not all rows were written to the binary log. (Bug#20821)
- Triggers on tables in the `mysql` database caused a server crash. Triggers for tables in this database now are disallowed. (Bug#18005, Bug#18361)
- The length of the pattern string prefix for `LIKE` operations was calculated incorrectly for multi-byte character sets. As a result, the scanned range was wider than necessary if the prefix contained any multi-byte characters, and rows could be missing from the result set. (Bug#16674, Bug#18359)
- Very complex `SELECT` statements could create temporary tables that were too big, but for which the temporary files did not get removed, causing subsequent queries to fail. (Bug#11824)
- Multiple-table updates with `FEDERATED` tables could cause a server crash. (Bug#19773)
- On Windows, terminating `mysqld` with Control-C could result in a crash during shutdown. (Bug#18235)
- Renaming a database to itself caused a server crash. (Bug#19392)
- For spatial data types, the server formerly returned these as `VARSTRING` values with a binary collation. Now the server returns spatial values as `BLOB` values. (Bug#10166)
- Using tables from MySQL 4.x in MySQL 5.x, in particular those with `VARCHAR` fields and using `INSERT DELAYED` to update data in the table would result in either data corruption or a server crash. (Bug#16611, Bug#16218, Bug#17294)
- Using `SELECT` and a table join while running a concurrent `INSERT` operation would join incorrect rows. (Bug#14400)
- Using `SELECT` on a corrupt `MyISAM` table using the dynamic record format could cause a server crash. (Bug#19835)
- Checking a `MyISAM` table (using `CHECK TABLE`) having a spatial index and only one row would wrongly indicate that the table was corrupted. (Bug#17877)
- A `TABLE ... DOESN'T EXIST` error could occur for statements that called a function defined in another database. (Bug#17199)
- `SHOW GRANTS FOR CURRENT_USER` did not return definer grants when executed in `DEFINER` context (such as within a stored procedure defined with `SQL SECURITY DEFINER`), it returned the invoker grants. (Bug#15298)
- Portions of statements related to partitioning were not surrounded by version-specific comments by `mysqldump`, breaking backwards compatibility for dump files. (Bug#19488)
- A `DELETE FROM table` with no `WHERE` clause (deleting all rows) running concurrently with `INSERT` statements on a storage engine with row-level locking (such as `NDB`) could produce inconsistent results when using statement-based replication. (Bug#19066)
- Concatenating the results of multiple constant subselects produced incorrect results. (Bug#16716)
- The use of `MIN()` and `MAX()` on columns with an index prefix produced incorrect results in some queries. (Bug#18206)
- The `WITH CHECK OPTION` was not enforced when a `REPLACE` statement was executed against a view. (Bug#19789)
- For `SELECT ... FOR UPDATE` statements that used `DISTINCT` or `GROUP BY` over all key parts of a unique index (or primary key), the optimizer unnecessarily created a temporary table, thus losing the linkage to the underlying unique index values. This caused a `Result set not updatable` error. (The temporary table is unnecessary because under these circumstances the distinct or grouped columns must also be unique.) (Bug#16458)
- A buffer overwrite error in Instance Manager caused a crash. (Bug#20622)
- Re-execution of a prepared multiple-table `DELETE` statement that involves a trigger or stored function can result in a server crash. (Bug#19634)

- On Windows, corrected a crash stemming from differences in Visual C runtime library routines from POSIX behavior regarding invalid file descriptors. ([Bug#18275](#))
- Creation of a view as a join of views or tables could fail if the views or tables are in different databases. ([Bug#20482](#))
- Use of `MIN()` or `MAX()` with `GROUP BY` on a `ucs2` column could cause a server crash. ([Bug#20076](#))
- `INSERT INTO ... SELECT ... LIMIT 1` could be slow because the `LIMIT` was ignored when selecting candidate rows. ([Bug#9676](#))
- Queries on tables that were partitioned by `KEY` and had a `VARCHAR` column as the partitioning key produced an empty result set. ([Bug#20086](#))
- The omission of leading zeros in dates could lead to erroneous results when these were compared with the output of certain date and time functions. ([Bug#16377](#))
- An invalid comparison between keys with index prefixes over multi-byte character fields could lead to incorrect result sets if the selected query execution plan used a range scan by an index prefix over a `UTF8` character field. This also caused incorrect results under similar circumstances with many other character sets. ([Bug#14896](#))
- A prepared statement that altered partitioned table within a stored procedure failed with the error `UNKNOWN PREPARED STATEMENT HANDLER`. ([Bug#17138](#))
- A query selecting records from a single partition of a partitioned table and using `ORDER BY ic DESC` (where `ic` represents an indexed column) could cause errors or crash the server. ([Bug#20583](#))
- **NDB Cluster** (Disk Data): On some platforms, `ndbd` compiled with `gcc 4` would crash when attempting to run `CREATE LOG-FILE GROUP`. ([Bug#21981](#))
- **NDB Cluster**: Setting `TransactionDeadlockDetectionTimeout` to a value greater than 12000 would cause scans to deadlock, time out, fail to release scan records, until the cluster ran out of scan records and stopped processing. ([Bug#21800](#))
- **NDB Cluster**: Data was stored unevenly between partitions due to all `BLOB` data being placed in partition 0. ([Bug#21690](#))
- **NDB Cluster**: The server provided a non-descriptive error message when encountering a fatally corrupted REDO log. ([Bug#21615](#))
- **NDB Cluster**: A partial rollback could lead to node restart failures. ([Bug#21536](#))
- **NDB Cluster**: The failure of a unique index read due to an invalid schema version could be handled incorrectly in some cases, leading to unpredictable results. ([Bug#21384](#))
- **NDB Cluster**: In a cluster with more than 2 replicas, a manual restart of one of the data nodes could fail and cause the other nodes in its nodegroup to shut down. ([Bug#21213](#))
- **NDB Cluster**: When the redo buffer ran out of space, a `POINTER TOO LARGE` error was raised and the cluster could become unusable until restarted with `--initial`. ([Bug#20892](#))
- **NDB Cluster**: In some situations with a high disk-load, writing of the redo log could hang, causing a crash with the error message `GCP STOP DETECTED`. ([Bug#20904](#))
- **NDB Cluster**: A vague error message was returned when reading of both schema files occurred during a restart of the cluster. ([Bug#20860](#))
- **NDB Cluster**: The server did not honor the value set for `ndb_cache_check_time` in the `my.cnf` file. ([Bug#20708](#))
- **NDB Cluster**: The server failed with a non-descriptive error message when out of data memory. ([Bug#18475](#))
- **NDB Cluster** (Direct APIss): Invoking the MGM API function `ndb_mgm_listen_event()` caused a memory leak. ([Bug#21671](#))
- **NDB Cluster**: The management client `ALL STATUS` command could sometimes report the status of some data nodes incorrectly. ([Bug#13985](#))
- **NDB Cluster** (Disk Data): Trying to create a Disk Data table using a nonexistent tablespace or trying to drop a nonexistent data file from a tablespace produced an uninformative error message. ([Bug#21751](#))

- **NDB Cluster** (Disk Data): Errors could occur when dropping a data file during a node local checkpoint. (Bug#21710)
- **NDB Cluster** (Disk Data): Creating a tablespace and log file group, then attempting to restart the cluster without using the `-initial` option and without having created any Disk Data tables could cause a forced shutdown of the cluster and raise a configuration error. (Bug#21172)
- **NDB Cluster**: Responses to the ALL DUMP 1000 management client command were printed multiple times in the cluster log for each cluster node. (Bug#21044)
- A memory leak was found when running `ndb_mgm -e "SHOW"`. (Bug#21670)
- **NDB Cluster** (Direct APIs): The MGM API function `ndb_logevent_get_fd()` was not actually implemented. (Bug#21129)
- **NDB Cluster**: Restarting a data node while DDL operations were in progress on the cluster could cause other data nodes to fail. This could also lead to `mysqld` hanging or crashing under some circumstances. (Bug#21017, Bug#21050)
- **NDB Cluster**: A cluster data node could crash when an ordered index became full before the table containing the index was full. (Bug#14935)
- **NDB Cluster**: The repeated creating and dropping of a table would eventually lead to NDB Error 826, `TOO MANY TABLES AND ATTRIBUTES ... INSUFFICIENT SPACE`. (Bug#20847)
- **NDB Cluster**: `REPLACE` statements did not work correctly on an NDB table having both a primary key and a unique key. In such cases, proper values were not set for columns which were not explicitly referenced in the statement. (Bug#20728)
- **NDB Cluster**: Trying to create or drop a table while a node was restarting caused the node to crash. This is now handled by raising an error. (Bug#18781)
- **NDB Cluster**: A race condition could in some circumstances following a `DROP TABLE`. (Bug#20897)
- **NDB Cluster**: Running `ndbd --nowait-nodes=id` where `id` was the node ID of a node that was already running would fail with an invalid error message. (Bug#20419)
- **NDB Cluster**: When stopping and restarting multiple data nodes, the last node to be restarted would sometimes hang in Phase 100. (Bug#19645)
- **NDB Cluster**: The `DATA_LENGTH` and `AVG_ROW_LENGTH` columns of the `INFORMATION_SCHEMA.TABLES` table did not report the size of variable-width column values correctly. (Bug#18413)

See Section 23.2, “The `INFORMATION_SCHEMA.TABLES` Table”, for more information.
- **NDB Cluster**: When attempting to restart the cluster following a data import, the cluster would fail during Phase 4 of the restart with `ERROR 2334: JOB BUFFER CONGESTION`. (Bug#20774)
- **NDB Cluster**: A node failure during a scan could sometime cause the node to crash when restarting too quickly following the failure. (Bug#20197)
- **NDB Cluster**: It was possible to use port numbers greater than 65535 for `ServerPort` in the `config.ini` file. (Bug#19164)
- **NDB Cluster** (Replication): In some cases, a large number of MySQL servers sending requests to the cluster simultaneously could cause the cluster to crash. This could also be triggered by many NDB API clients making simultaneous event subscriptions or unsubscriptions. (Bug#20683)
- **NDB Cluster** (Direct APIs): `NdbScanOperation::readTuples()` and `NdbIndexScanOperation::readTuples()` ignored the `batch` parameter. (Bug#20252)
- **NDB Cluster**: Cluster system status variables were not updated. (Bug#11459)
- **NDB Cluster** (Disk Data): Trying to create Disk Data tables when running the cluster in diskless mode would crash the cluster's data nodes. (Bug#20008)

Note

Disk Data tables are now disabled when running in diskless mode.

- **NDB Cluster** (Disk Data): A data file created on one tablespace could be dropped using `ALTER TABLESPACE ... DROP DATAFILE` on a different tablespace. (Bug#20053)
- **NDB Cluster**: Truncating a table on one `mysqld` caused other `mysqld` processes in the cluster to return `ERROR 1412 (HY000): TABLE DEFINITION HAS CHANGED, PLEASE RETRY TRANSACTION` on subsequent queries. (Bug#20705)
- **NDB Cluster**: The cluster's data nodes would fail while trying to load data when `NoOfFrangmentLogFiles` was equal to 1. (Bug#19894)
- **NDB Cluster**: A problem with error handling when `ndb_use_exact_count` was enabled could lead to incorrect values returned from queries using `COUNT()`. A warning is now returned in such cases. (Bug#19202)
- **NDB Cluster**: Restarting a failed node could crash the cluster. (Bug#18782)
- **NDB Cluster** (Disk Data): The failure of a `CREATE TABLESPACE` or `CREATE LOGFILE GROUP` statement did not revert all changes made prior to the point of failure. (Bug#16341)
- **NDB Cluster**: Creating tables with variable-size columns caused `DataMemory` to be used but not freed when the tables were dropped. (Bug#20007)
- **NDB Cluster**: Restoring a backup made using `ndb_restore` failed when the backup had been taken from a cluster whose data memory was full. (Bug#19852)
- **NDB Cluster**: An excessive number of `ALTER TABLE` operations could cause the cluster to fail with NDB error code 773 (`OUT OF STRING MEMORY, PLEASE MODIFY STRINGMEMORY`). (Bug#19275)
- **NDB Cluster**: `TEXT` columns in Cluster tables having both an explicit primary key and a unique key were not correctly updated by `REPLACE` statements. (Bug#19906)
- **NDB Cluster**: Running out of `DataMemory` could sometimes crash `ndbd` and `mysqld` processes. (Bug#19185)
- **NDB Cluster** (Replication): A node failure could send duplicate events, causing a `mysqld` replicating tables containing `BLOBs` to crash.
- **NDB Cluster** (Disk Data): `INFORMATION_SCHEMA.FILES` records for `UNDO` files showed incorrect values in the `EXTENT_SIZE`, `FREE_EXTENTS`, and `TOTAL_EXTENTS` columns. (Bug#20073)
- **NDB Cluster**: An internal formatting error caused some management client error messages to be unreadable. (Bug#20016)
- **NDB Cluster**: Running management client commands while `mgmd` was in the process of disconnecting could cause the management server to fail. (Bug#19932)
- **NDB Cluster** (NDBAPI): Update operations on blobs were not checked for illegal operations.

Note

Read locks with blob update operations are now upgraded from read committed to read shared.

- **NDB Cluster**: The management client `ALL STOP` command shut down `mgmd` processes (as well as `ndbd` processes). (Bug#18966)
- **NDB Cluster**: Renaming of table columns was not supported as fast a `ALTER TABLE` for NDB tables. (Bug#20456)
- **NDB Cluster**: Under some circumstances, repeated DDL operations on one `mysqld` could cause failure of a second `mysqld` attached to the same cluster. (Bug#19770)
- **NDB Cluster** (Replication): One or more of the `mysqld` processes could fail when subjecting a Cluster replication setup with multiple `mysqld` processes on both the master and slave clusters to high loads. (Bug#19768)
- **NDB Cluster**: `LOAD DATA LOCAL` failed to ignore duplicate keys in Cluster tables. (Bug#19496)
- **NDB Cluster**: Repeated `CREATE - INSERT - DROP` operations tables could in some circumstances cause the MySQL table definition cache to become corrupt, so that some `mysqld` processes could access table information but others could not. (Bug#18595)
- **NDB Cluster** (Disk Data): Running a large number of scans on Disk Data could cause subsequent scans to perform poorly.

[Bug#20334](#))

- **NDB Cluster** (Disk Data): An issue with disk allocation could sometimes cause a forced shutdown of the cluster when running a mix of memory and Disk Data tables. ([Bug#18780](#))
- **NDB Cluster**: The `mgm` client command `ALL CLUSTERLOG STATISTICS=15;` had no effect. ([Bug#20336](#))
- **NDB Cluster**: Under certain conditions, a starting node could miss transactions, leading to inconsistencies between the primary and backup replicas. ([Bug#19929](#))
- **NDB Cluster**: An uncommitted row could sometimes be checkpointed and thus incorrectly included in a backup. ([Bug#19928](#))
- **NDB Cluster**: A `DELETE` of many rows immediately followed by an `INSERT` on the same table could cause the `ndbd` process on the backup replica to crash. ([Bug#19293](#))
- **NDB Cluster**: `TRUNCATE TABLE` failed to reset the `AUTO_INCREMENT` counter. ([Bug#18864](#))
- **NDB Cluster**: `SELECT ... FOR UPDATE` failed to lock the selected rows. ([Bug#18184](#))
- **NDB Cluster**: New `mysqld` processes were allowed to connect without a restart of the cluster, causing the cluster to crash. ([Bug#13266](#))
- **NDB Cluster**: The failure of a data node when preparing to commit a transaction (that is, while the node's status was `CS_PREPARE_TO_COMMIT`) could cause the failure of other cluster data nodes. ([Bug#20185](#))
- **NDB Cluster**: Renaming a table in such a way as to move it to a different database failed to move the table's indexes. ([Bug#19967](#))
- **NDB Cluster**: `SHOW ENGINE NDB STATUS` could sometimes return an incorrect value of `0` for the latest epoch, which could cause problems with synchronising the binlog. ([Bug#20142](#))
- **NDB Cluster**: A `CREATE TABLE` statement involving foreign key constraints raised an error rather than being silently ignored (see [Section 12.1.10, "CREATE TABLE Syntax"](#)). ([Bug#18483](#))

This bug affected Cluster in MySQL 5.1 only.

- **NDB Cluster** (Replication): Data definition and data manipulation statements on different tables were not serialised correctly in the binlog. For example, there was no guarantee that a `CREATE TABLE` statement and an update on a different table would occur in the same order in the binlog as they did on the cluster being replicated. ([Bug#18947](#))
- **NDB Cluster**: Resources for unique indexes on Cluster table columns were incorrectly allocated, so that only one-fourth as many unique indexes as indicated by the value of `UniqueHashIndexes` could be created. ([Bug#19623](#))
- A cast problem caused incorrect results for prepared statements that returned float values when MySQL was compiled with `gcc 4.0`. ([Bug#19694](#))
- Some queries that used `ORDER BY` and `LIMIT` performed quickly in MySQL 3.23, but slowly in MySQL 4.x/5.x due to an optimizer problem. ([Bug#4981](#))
- `mysql_upgrade` was missing from binary MySQL distributions. ([Bug#18516](#), [Bug#20403](#), [Bug#20556](#))
- It was possible using `ALTER EVENT ... RENAME ...` to move an event to a database on which the user did not have the `EVENT` privilege. ([Bug#18897](#))
- A number of dependency issues in the RPM `bench` and `test` packages caused installation of these packages to fail. ([Bug#20078](#))
- Queries using an indexed column as the argument for the `MIN()` and `MAX()` functions following an `ALTER TABLE ... DISABLE KEYS` statement returned `GOT ERROR 124 FROM STORAGE ENGINE` until `ALTER TABLE ... ENABLE KEYS` was run on the table. ([Bug#20357](#))
- A redundant table map event could be generated in the binary log when there were no actual changes to a table being replicated. In addition, a slave failed to stop when attempting to replicate a table that did not exist on the slave. ([Bug#18948](#))
- Multiple calls to a stored procedure that altered a partitioned `MyISAM` table would cause the server to crash. ([Bug#19309](#))
- Adding an index to a partitioned table that had been created using `AUTO_INCREMENT = value` caused the `AUTO_INCREMENT`

value to be reset. ([Bug#19281](#))

- A `CREATE TABLE` that produced a `THE PARTITION FUNCTION RETURNS THE WRONG TYPE` error also caused an `INCORRECT INFORMATION IN FILE` to be printed to `STDERR`, and a junk file to be left in the database directory. ([Bug#16000](#))
- Nested natural joins worked executed correctly when executed as a non-prepared statement could fail with an `Unknown column 'col_name' in 'field list'` error when executed as a prepared statement, due to a name resolution problem. ([Bug#15355](#))
- The `max_length` metadata value for columns created from `CONCAT()` could be incorrect when the collation of an argument differed from the collation of the `CONCAT()` itself. In some contexts such as `UNION`, this could lead to truncation of the column contents. ([Bug#15962](#))
- The MD5() and SHA() functions treat their arguments as case-sensitive strings. But when they are compared, their arguments were compared as case-insensitive strings, which leads to two function calls with different arguments (and thus different results) compared as being identical. This can lead to a wrong decision made in the range optimizer and thus to an incorrect result set. ([Bug#15351](#))
- For `BOOLEAN` mode full-text searches on non-indexed columns, `NULL` rows generated by a `LEFT JOIN` caused incorrect query results. ([Bug#14708](#))
- If the general log table reached a large enough file size (27GB), `SELECT COUNT(*)` on the table caused a server crash. ([Bug#17589](#))
- Identifiers could not contain bytes with a value of 255, though that should be allowed as of the identifier-encoding changes made in MySQL 5.1.6. ([Bug#12982](#))
- `BIT` columns in a table could cause joins that use the table to fail. ([Bug#18895](#))
- A `UNION` over more than 128 `SELECT` statements that use an aggregate function failed. ([Bug#18175](#))
- `InnoDB` unlocked its data directory before committing a transaction, potentially resulting in non-recoverable tables if a server crash occurred before the commit. ([Bug#19727](#))
- Multiple-table `DELETE` statements containing a subquery that selected from one of the tables being modified caused a server crash. ([Bug#19225](#))
- With settings of `read_buffer_size` $\geq$ 2G and `read_rnd_buffer_size` $\geq$ 2G, `LOAD DATA INFILE` failed with no error message or caused a server crash for files larger than 2GB. ([Bug#12982](#))
- `REPLACE` statements caused activation of `UPDATE` triggers, not `DELETE` and `INSERT` triggers. ([Bug#13479](#))
- The thread for `INSERT DELAYED` rows was maintaining a separate `AUTO_INCREMENT` counter, resulting in incorrect values being assigned if `DELAYED` and non-`DELAYED` inserts were mixed. ([Bug#20195](#))
- `mysqldump` wrote an extra pair of `DROP DATABASE` and `CREATE DATABASE` statements if run with the `-add-drop-database` option and the database contained views. ([Bug#17201](#))
- Shutting down a slave in a replication scenario where temporary tables are in use would cause the slave to produce a core dump. ([Bug#19881](#))
- When a statement is executed that does not generate any rows, an extra table map event and associated binrows event would be generated and written to the binary log. ([Bug#19995](#))
- File size specifications for `InnoDB` data files were case sensitive. ([Bug#19609](#))
- Compilation on Windows would fail if row based replication was disabled using `--without-row-based-replication`. ([Bug#16837](#))
- `InnoDB` did not increment the `handler_read_prev` counter. ([Bug#19542](#))
- In the `INFORMATION_SCHEMA.FILES` table, the `INITIAL_SIZE`, `MAXIMUM_SIZE`, and `AUTOEXTEND_SIZE` columns incorrectly were being stored as `VARCHAR` rather than `BIGINT`. ([Bug#19544](#))
- For `mysqld`, Valgrind revealed problems that were corrected: Possible uninitialized data in a string comparison ([Bug#20783](#)); memory corruption in replication slaves when switching databases ([Bug#19022](#)); syscall write parameter pointing to uninitialized

byte ([Bug#20579](#)); uninitialized doublewrite memory in [InnoDB](#) ([Bug#20791](#)).

- For [ndb_mgmd](#), Valgrind revealed problems that were corrected: A memory leak ([Bug#19318](#)); a dependency on an uninitialized variable ([Bug#20333](#)).
- An update that used a join of a table to itself and modified the table on both sides of the join reported the table as crashed. ([Bug#18036](#))
- SSL connections using yaSSL on OpenBSD could fail. ([Bug#19191](#))
- Following a failed attempt to add an index to an [ARCHIVE](#) table, it was no longer possible to drop the database in which the table had been created. ([Bug#17310](#))
- Using `ALTER TABLE ... ENGINE = x`, where `x` was not a storage engine supported by the server, would cause `mysqld` to crash. ([Bug#20397](#))
- Defining a table partitioned by [LIST](#) with a single `PARTITION ... VALUES IN (NULL)` clause could lead to server crashes, particularly with queries having `WHERE` conditions comparing the partitioning key with a constant. ([Bug#19801](#) / [Bug#20268](#))
- Values greater than 2 gigabytes used in the `VALUES LESS THAN` clause of a table partitioned by `RANGE` were treated as negative numbers. ([Bug#16002](#))
- The `fill_help_tables.sql` file did not load properly if the `ANSI_QUOTES` SQL mode was enabled. ([Bug#20542](#))
- The `fill_help_tables.sql` file did not contain a `SET NAMES 'utf8'` statement to indicate its encoding. This caused problems for some settings of the MySQL character set such as `big5`. ([Bug#20551](#))
- The `--default-storage-engine` server option did not work. ([Bug#20168](#))
- The MySQL server startup script `/etc/init.d/mysql` (created from `mysql.server`) is now marked to ensure that the system services `yppbind`, `nscd`, `ldap`, and `NTP` are started first (if these are configured on the machine). ([Bug#18810](#))
- For a reference to a non-existent index in `FORCE INDEX`, the error message referred to a column, not an index. ([Bug#17873](#))
- The `ENGINE` clause was displayed in the output of `SHOW CREATE TABLE` for partitioned tables when the SQL mode included `no_table_options`. ([Bug#19695](#))
- `ALTER TABLE ... COALESCE PARTITION` did not delete the files associated with the partitions that were removed. ([Bug#19305](#))
- `ALTER TABLE ... REBUILD PARTITION` could cause the server to hang or crash. ([Bug#19122](#))
- Some yaSSL public function names conflicted with those from OpenSSL, causing conflicts for applications that linked against both OpenSSL and a version of `libmysqlclient` that was built with yaSSL support. The yaSSL public functions now are renamed to avoid this conflict. ([Bug#19575](#))
- `CHECK TABLE` on a `MyISAM` table briefly cleared its `AUTO_INCREMENT` value, while holding only a read lock. Concurrent inserts to that table could use the wrong `AUTO_INCREMENT` value. `CHECK TABLE` no longer modifies the `AUTO_INCREMENT` value. ([Bug#19604](#))
- If there is a global read lock, `CREATE DATABASE`, `RENAME DATABASE`, and `DROP DATABASE` could deadlock. ([Bug#19815](#))
- `EXPLAIN PARTITIONS` would produce illegible output in the `partitions` column if the length of text to be displayed in that column was too long. This could occur when very many partitions were defined for the table, partitions were given very long names, or due to a combination of the two. ([Bug#19684](#))
- Trying to execute a query having a `WHERE` clause using `int_col = "string_value" OR int_col IS NULL` on a partitioned table whose partitioning or subpartitioning function used the integer column `int_col` would crash the server. ([Bug#19055](#))
- On Linux, `libmysqlclient` when compiled with yaSSL using the `icc` compiler had a spurious dependency on C++ libraries. ([Bug#20119](#))
- In MySQL 5.1.11, the `--with-openssl` and `--with-yassl` options were replaced by `--with-ssl`. But no message was issued if the old options were given. Now `configure` produces a message indicating that the new option should be used and exits. ([Bug#20002](#))

- Grant table modifications sometimes did not refresh the in-memory tables if the hostname was ' ' or not specified. (Bug#16297)
- Invalid escape sequences in option files caused MySQL programs that read them to abort. (Bug#15328)
- Using `ALTER TABLE` on a subpartitioned table caused the server to crash. (Bug#19067)
- For a table having `LINEAR HASH` subpartitions, the `LINEAR` keyword did not appear in the `SUBPARTITION_METHOD` column of the `INFORMATION_SCHEMA.PARTITIONS` table. (Bug#20161)
- Race conditions on certain platforms could cause the Instance Manager to fail to initialize. (Bug#19391)
- `ALTER TABLE` on a table created prior to 5.0.3 would cause table corruption if the `ALTER TABLE` did one of the following:
 - Change the default value of a column.
 - Change the table comment.
 - Change the table password.(Bug#17001)
- An `ALTER TABLE` operation that does not need to copy data, when executed on a table created prior to MySQL 4.0.25, could result in a server crash for subsequent accesses to the table. (Bug#19192)
- The binary log lacked character set information for table name when dropping temporary tables. (Bug#14157)
- A `B-TREE` index on a `MEMORY` table erroneously reported duplicate entry error for multiple `NULL` values. (Bug#12873)
- Race conditions on certain platforms could cause the Instance Manager to try to restart the same instance multiple times. (Bug#18023)
- `OPTIMIZE TABLE` and `REPAIR TABLE` yielded incorrect messages or warnings when used on partitioned tables. (Bug#17455)
- Selecting data from a `MEMORY` table with a `VARCHAR` column and a `HASH` index over it returned only the first row matched. (Bug#18233)
- RPM packages had spurious dependencies on Perl modules and other programs. (Bug#13634)

C.1.19. Changes in release 5.1.11 (26 May 2006)

This is a new Beta development release, fixing recently discovered bugs.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- **Incompatible change:** The Event Scheduler can now be in one of three states (on, off, or the new suspended state). In addition, due to the fact that `SET GLOBAL event_scheduler;` now acts in a synchronous rather than asynchronous manner, the Event Scheduler thread can no longer be activated or deactivated at run time. (Bug#17619)

For more information regarding these changes, see [Section 21.1, “Event Scheduler Overview”](#).

- Previously, to build MySQL from source with SSL support enabled, you would invoke `configure` with either the `--with-openssl` or `--with-yassl` option. Those options both have been replaced by the `--with-ssl` option. By default, `--with-ssl` causes the bundled yaSSL library to be used. To select OpenSSL instead, give the option as `--with-ssl=path`, where `path` is the directory where the OpenSSL header files and libraries are located.
- Added the `--ssl-verify-server-cert` option to MySQL client programs. This option causes the server's Common Name value in its certificate to be verified against the hostname used when connecting to the server, and the connection is rejected if there is a mismatch. Added `MYSQL_OPT_SSL_VERIFY_SERVER_CERT` option for the `mysql_options()` C API function to enable this verification. This feature can be used to prevent man-in-the-middle attacks. Verification is disabled by default.

)

- Added the `ssl_ca`, `ssl_capath`, `ssl_cert`, `ssl_cipher`, and `ssl_key` system variables, which display the values given via the corresponding command options. See [Section 5.8.7.3, “SSL Command Options”](#). ([Bug#19606](#))
- **NDB Cluster**: The limit of 2048 ordered indexes per cluster has been lifted. There is now no upper limit on the number of ordered indexes (including `AUTO_INCREMENT` columns) that may be used. ([Bug#14509](#))
- Added the `log_queries_not_using_indexes` system variable. ([Bug#19616](#))
- Added the `--angel-pid-file` option to `mysqlmanager` for specifying the file in which the angel process records its process ID when `mysqlmanager` runs in daemon mode. ([Bug#14106](#))
- The `ENABLE KEYS` and `DISABLE KEYS` clauses for the `ALTER TABLE` statement are now supported for partitioned tables. ([Bug#19502](#))
- It is now possible to use `NEW.var_name` values within triggers as `INOUT` parameters to stored procedures. ([Bug#14635](#))
- The default for the `innodb_thread_concurrency` system variable was changed to 8. ([Bug#15868](#))
- `mysql_explain_log` (a third-party program) is no longer included in MySQL distributions.

Bugs fixed:

- **Security fix**: An SQL-injection security hole has been found in multi-byte encoding processing. The bug was in the server, incorrectly parsing the string escaped with the `mysql_real_escape_string()` C API function. ([CVE-2006-2753](#), [Bug#8378](#))

This vulnerability was discovered and reported by Josh Berkus <josh@postgresql.org> and Tom Lane <tgl@sss.pgh.pa.us> as part of the inter-project security collaboration of the OSDB consortium. For more information about SQL injection, please see the following text.

Discussion: An SQL-injection security hole has been found in multi-byte encoding processing. An SQL-injection security hole can include a situation whereby when a user supplied data to be inserted into a database, the user might inject SQL statements into the data that the server will execute. With regards to this vulnerability, when character set unaware-escaping is used (for example, `addslashes()` in PHP), it is possible to bypass the escaping in some multi-byte character sets (for example, SJIS, BIG5 and GBK). As a result, a function such as `addslashes()` is not able to prevent SQL-injection attacks. It is impossible to fix this on the server side. The best solution is for applications to use character set-aware escaping offered by a function such `mysql_real_escape_string()`.

However, a bug was detected in how the MySQL server parses the output of `mysql_real_escape_string()`. As a result, even when the character set-aware function `mysql_real_escape_string()` was used, SQL injection was possible. This bug has been fixed.

Workarounds: If you are unable to upgrade MySQL to a version that includes the fix for the bug in `mysql_real_escape_string()` parsing, but run MySQL 5.0.1 or higher, you can use the `NO_BACKSLASH_ESCAPES` SQL mode as a workaround. (This mode was introduced in MySQL 5.0.1.) `NO_BACKSLASH_ESCAPES` enables an SQL standard compatibility mode, where backslash is not considered a special character. The result will be that queries will fail.

To set this mode for the current connection, enter the following SQL statement:

```
SET sql_mode='NO_BACKSLASH_ESCAPES';
```

You can also set the mode globally for all clients:

```
SET GLOBAL sql_mode='NO_BACKSLASH_ESCAPES';
```

This SQL mode also can be enabled automatically when the server starts by using the command-line option `-sql-mode=NO_BACKSLASH_ESCAPES` or by setting `sql-mode=NO_BACKSLASH_ESCAPES` in the server option file (for example, `my.cnf` or `my.ini`, depending on your system).

- The patch for [Bug#8303](#) broke the fix for [Bug#8378](#) and was undone. (In string literals with an escape character (`\`) followed by a multi-byte character that has a second byte of (`\`), the literal was not interpreted correctly. The next byte now is escaped, not the entire multi-byte character. This means it a strict reverse of the `mysql_real_escape_string()` function.)

- The client libraries had not been compiled for position-independent code on Solaris-SPARC and AMD x86_64 platforms. ([Bug#13159](#), [Bug#14202](#), [Bug#18091](#))
 - Altering a `VARCHAR` column in a `MyISAM` table to make it longer could cause corruption of the following column. ([Bug#19386](#))
 - A `CREATE TABLE` statement that created a table from a materialized view did not inherit default values from the underlying table. ([Bug#19089](#))
 - **NDB Cluster**: A Cluster whose storage nodes were installed from the `MySQL-ndb-storage-*` RPMs could not perform `CREATE` or `ALTER` operations that made use of non-default character sets or collations. ([Bug#14918](#))
 - **NDB Cluster**: `mysqld` processes did not always detect cluster shutdown, leading to issues with Cluster replication and schema distribution. ([Bug#19395](#))
 - **NDB Cluster**: `SELECT MIN(unique_column)` from a Cluster table with user-defined partitioning crashed the server. ([Bug#18730](#))
 - Premature optimization of nested subqueries in the `FROM` clause that refer to aggregate functions could lead to incorrect results. ([Bug#19077](#))
 - For dates with 4-digit year parts less than 200, an implicit conversion to add a century was applied for date arithmetic performed with `DATE_ADD()`, `DATE_SUB()`, `+ INTERVAL`, and `- INTERVAL`. (For example, `DATE_ADD('0050-01-01 00:00:00', INTERVAL 0 SECOND)` became `'2050-01-01 00:00:00'`.) Now these operations return `NULL` rather than an incorrect non-`NULL` value. ([Bug#18997](#))
 - `BLOB` or `TEXT` arguments to or values returned from stored functions were not copied properly if too long and could become garbled. ([Bug#18587](#))
 - Simultaneous scheduled events whose actions conflicted with one another could crash the server. ([Bug#16428](#))
 - It was not possible to invoke a stored routine containing dynamic SQL from a scheduled event. ([Bug#19264](#))
 - **NDB Cluster**: Running `ALL START` in the **NDB** management client or restarting multiple nodes simultaneously could under some circumstances cause the cluster to crash. ([Bug#19930](#))
 - The result from `CONV()` is a string, but was not always treated the same way as a string when converted to a real value for an arithmetic operation. ([Bug#13975](#))
 - `CREATE TABLE ... SELECT ...` statements that used a stored function explicitly or implicitly (through a view) resulted in a `Table not locked` error. ([Bug#12472](#), [Bug#15137](#))
 - Within a trigger, `SET` used the SQL mode of the invoking statement, not the mode in effect at trigger creation time. ([Bug#6951](#))
 - The server no longer uses a signal handler for signal 0 because it could cause a crash on some platforms. ([Bug#15869](#))
 - The embedded server crashed with row-based replication enabled. ([Bug#18518](#))
 - Display better error message for `ALTER TABLE` operations that will result in duplicate keys due to `AUTO_INCREMENT` resequencing. ([Bug#14573](#))
 - The `Data_free` column in the output of `SHOW TABLE STATUS` always displayed 0 for partitioned tables. ([Bug#19501](#))
 - Adding an index to a table created using partitioning by `KEY` and the `MEMORY` storage engine caused the server to crash. ([Bug#19140](#))
 - When creating a table using `CREATE TABLE ... PARTITION BY ... SELECT ...`, the partitioning clause was ignored. ([Bug#19062](#))
 - `ALTER TABLE ENGINE=...` failed when used to change a MySQL Cluster table having no explicit primary key to use a different storage engine. ([Bug#19010](#))
- Note:* As a consequence of this fix, `SHOW CREATE TABLE` no longer displays auto-partitioning information for **NDBCluster** tables.
- **NDB Cluster** (NDBAPI): On big-endian platforms, `NdbOperation::write_attr()` did not update 32-bit fields correctly. ([Bug#19537](#))

- **NDB Cluster**: Using “stale” `mysqld .FRM` files could cause a newly-restored cluster to fail. This situation could arise when re-starting a MySQL Cluster using the `--initial` option while leaving connected `mysqld` processes running. (Bug#16875)
- **NDB Cluster** (Replication): Memory was not freed after some `ALTER TABLE` operations, which could cause `mysqld` processes to crash. (Bug#19885)
- **NDB Cluster** (NDBAPI): The `Ndb::dropEventOperation()` method failed to clean up all objects used, which could cause memory leaks to occur. (Bug#17610)
- **NDB Cluster**: Data node failures could cause excessive CPU usage by `ndb_mgmd`. (Bug#13987)
- **NDB Cluster**: `TRUNCATE` failed on tables having `BLOB` or `TEXT` columns with the error `LOCK WAIT TIMEOUT EXCEEDED`. This affected both in-memory and Disk Data tables. (Bug#19201)
- Revised memory allocation for local objects within stored functions and triggers to avoid memory leak for repeated function or trigger invocation. (Bug#17260)
- `EXPLAIN ... SELECT INTO` caused the client to hang. (Bug#15463)
- Symlinking `.mysql_history` to `/dev/null` to suppress statement history saving by `mysql` did not work. (`mysql` deleted the symlink and recreated `.mysql_history` as a regular file, and then wrote history to it.) (Bug#16803)
- The `basedir` and `tmpdir` system variables could not be accessed via `@@var_name` syntax. (Bug#1039)
- Corrected several problems with the treatment of the `--log-error` option by `mysqld_safe`. These problems were manifest as differences from `mysqld` in error log handling.
 - If a filename was given for `--log-error`, `mysqld_safe` ignored it and did not pass it to `mysqld`, which then wrote error information to `stderr` and resulted in incorrect log rotation when `FLUSH LOGS` was used.
 - `mysqld_safe` now adds `.err` to the end of the filename if no extension is present (the same as `mysqld`).
 - `mysqld_safe` treated a relative pathname as relative to its own current working directory. Now it treats a relative pathname as relative to the data directory (the same as `mysqld`).

In addition, some argument quoting problems were corrected. (Bug#6061)

- Returning the value of a system variable from a stored function caused a server crash. (Bug#18037)
- Use of uninitialized user variables in a subquery in the `FROM` clause resulted in bad entries in the binary log. (Bug#19136)
- `IS_USED_LOCK()` could return an incorrect connection identifier. (Bug#16501)
- Concurrent reading and writing of privilege structures could crash the server. (Bug#16372)

C.1.20. Changes in release 5.1.10 (Not released)

Note

This was an internal release only, and no binaries were published.

MySQL 5.1.10 includes the patches for recently reported security vulnerabilities in the MySQL client-server protocol. We would like to thank Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting these to us.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- **Security enhancement**: Added the global `max_prepared_stmt_count` system variable to limit the total number of prepared statements in the server. This limits the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. The current number of prepared statements is available through the `prepared_stmt_count`

system variable. (Bug#16365)

- The `mysql_upgrade` command has been converted from a shell script to a C program, so it is available on non-Unix systems such as Windows. This program should be run for each MySQL upgrade. See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).
- Binary distributions that include SSL support now are built using yaSSL when possible.
- The `MySQL-shared-compat-5.1.X-.i386.rpm` shared compatibility RPMs no longer contain libraries for MySQL 5.0. This avoids a conflict because the 5.0 and 5.1 libraries share the same soname number. It contains libraries for 3.23, 4.0, 4.1, and 5.1. (Bug#19288)
- The `ONLY_FULL_GROUP_BY` SQL mode now also applies to the `HAVING` clause. That is, columns not named in the `GROUP BY` clause cannot be used in the `HAVING` clause if not used in an aggregate function. (Bug#18739)
- SQL syntax for prepared statements now supports `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE`. (Bug#19308)
- XPath expressions passed to the `ExtractValue()` and `UpdateXML()` functions can now include the colon character (“ : ”). This enables use of these functions with XML which employs namespaces. (Bug#18170)
- The bundled yaSSL library was upgraded to version 1.3.5. This improves handling of certain problems with SSL-related command options. (Bug#17737)
- For the `mysql` client, typing Control-C causes `mysql` to attempt to kill the current statement. If this cannot be done, or Control-C is typed again before the statement is killed, `mysql` exits. Previously, Control-C caused `mysql` to exit in all cases. (Bug#1989)
- Creating a table in an InnoDB database with a column name that matched the name of an internal InnoDB column (including `DB_ROW_ID`, `DB_TRX_ID`, `DB_ROLL_PTR` and `DB_MIX_ID`) would cause a crash. MySQL now returns error 1005 (cannot create table) with `errno` set to -1. (Bug#18934)
- On Windows, some names such as `nul`, `prn`, and `aux` could not be used as filenames because they are reserved as device names. These are now allowable names in MySQL. They are encoded by appending `@@@` to the name when the server creates the corresponding file or directory. This occurs on all platforms for portability of the corresponding database object between platforms. (Bug#17870)
- Added the `REFERENTIAL_CONSTRAINTS` table to `INFORMATION_SCHEMA`. It provides information about foreign keys.
- Added `--debug` option to Instance Manager.
- Added the `have_dynamic_loading` system variable that indicates whether the server supports dynamic loading of plugins.
- Added the `sql_big_selects` system variable to the output of `SHOW VARIABLES`.
- You must now have the `DROP` privilege to drop table partitions. (Bug#17139)
- **NDB Cluster**: It is now possible to perform a partial start of a cluster. That is, it is now possible to bring up the cluster without running `ndbd --initial` on all configured data nodes first. (Bug#18606)
- **NDB Cluster**: It is now possible to restore a Cluster backup between big-endian and little-endian machines. (Bug#19255)
- **NDB Cluster**: It is now possible to install MySQL with Cluster support to a non-default location and change the search path for font description files using either the `--basedir` or `--character-sets-dir` options. (Previously in MySQL 5.1, `ndbd` searched only the default path for character sets.)
- In result set metadata, the `MYSQL_FIELD.length` value for `BIT` columns now is reported in number of bits. For example, the value for a `BIT(9)` column is 9. (Formerly, the value was related to number of bytes.) (Bug#13601)
- Added the `KEY_BLOCK_SIZE` table option and index option. This can be used in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements to provide a hint to the storage engine about the size to use for index key blocks. The engine is allowed to change the value if necessary.

Bugs fixed:

- **Security fix**: A malicious client, using specially crafted invalid login or `COM_TABLE_DUMP` packets was able to read uninitialized

memory, which potentially, though unlikely in MySQL, could have led to an information disclosure. (CVE-2006-1516, CVE-2006-1517) Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting this bug.

- **Security fix:** A malicious client, using specially crafted invalid `COM_TABLE_DUMP` packets was able to trigger an exploitable buffer overflow on the server. (CVE-2006-1518) Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting this bug.
- **NDB Cluster** (Replication): Using the `--binlog-do-db` option caused problems with `CREATE TABLE` on the cluster acting as the replication master. (Bug#19492)
- **NDB Cluster:** Some queries having a `WHERE` clause of the form `c1=val1 OR c2 LIKE 'val2'` were not evaluated correctly. (Bug#17421)
- **NDB Cluster:** Repeated use of the `SHOW` and `ALL STATUS` commands in the `ndb_mgm` client could cause the `mgmd` process to crash. (Bug#18591)
- **NDB Cluster:** An issue with `ndb_mgmd` prevented more than 27 `mysqld` processes from connecting to a single cluster at one time. (Bug#17150)
- Running `myisampack` followed by `myisamchk` with the `--unpack` option would corrupt the `auto_increment` key. (Bug#12633)
- A view definition that referred to an alias in the `HAVING` clause could be saved in the `.frm` file with the alias replaced by the expression that it referred to, causing failure of subsequent `SELECT * FROM view_name` statements. (Bug#19573)
- A compatibility issue with NPTL (Native POSIX Thread Library) on Linux could result in a deadlock with `FLUSH TABLES WITH READ LOCK` under some conditions. (Bug#20048)
- **MyISAM** table deadlock was possible if one thread issued a `LOCK TABLES` request for write locks and then an administrative statement such as `OPTIMIZE TABLE`, if between the two statements another client meanwhile issued a multiple-table `SELECT` for some of the locked tables. (Bug#16986)
- The patch for Bug#17164 introduced the problem that some outer joins were incorrectly converted to inner joins. (Bug#19816)
- A `NUL` byte within a comment in a statement string caused the rest of the string not to be written to the query log, allowing logging to be bypassed. (CVE-2006-0903) (Bug#17667)
- **NDB Cluster:** `mysqld` could crash when attempting an update if the cluster had failed previously. (Bug#18798)
- `mysql-test-run.pl` started **NDB** even for test cases that didn't need it. (Bug#19083)
- Selecting from a view that used `GROUP BY` on a non-constant temporal interval (such as `DATE(col) + INTERVAL TIME_TO_SEC(col) SECOND`) could cause a server crash. (Bug#19490)
- An outer join of two views that was written using `{ OJ ... }` syntax could cause a server crash. (Bug#19396)
- `mysql` displayed `NULL` for strings that are empty or contain only spaces. (Bug#19564)
- A range access optimizer heuristic was invalid, causing some queries to be much slower in MySQL 5.0 than in 4.0. (Bug#17379, Bug#18940)
- `SELECT DISTINCT` queries sometimes returned only the last row. (Bug#18068)
- Eliminated some memory corruption problems that resulted in `double free or corruption` errors and a server crash. (Bug#19154)
- Use of `CONVERT_TZ()` in a stored function or trigger (or in a stored procedure called from a stored function or trigger) caused an error. (Bug#11081)
- Some queries were slower in 5.0 than in 4.1 because some 4.1 cost-evaluation code had not been merged into 5.0. (Bug#14292)
- `MySQL-shared-compat-5.1.9-0.i386.rpm` incorrectly depended on `glibc` 2.3 and could not be installed on a `glibc` 2.2 system. (Bug#16539)
- Updates to a **MEMORY** table caused the size of **BTREE** indexes for the table to increase. (Bug#18160)

- `REPAIR TABLE` did not restore the length for packed keys in tables created under MySQL 4.x. (Bug#17810)
- The parser leaked memory when its stack needed to be extended. (Bug#18930)
- When `myisamchk` needed to rebuild a table, `AUTO_INCREMENT` information was lost. (Bug#10405)
- Use of `--default-storage-engine=innodb` resulted in an error with the server reporting that `InnoDB` was an unknown table type. (Bug#16691)
- Executing a `CREATE EVENT` statement could cause 100% CPU usage. (Bug#19170)
- After calling `FLUSH STATUS`, the `max_used_connections` variable did not increment for existing connections and connections which use the thread cache. (Bug#15933)
- MySQL would not compile on Linux distributions that use the tinfo library. (Bug#18912)
- An issue with file handling in the partitioning code could cause `mysqld` to crash when started and then stopped within a very short period of time. (Bug#19313)
- **NDB Cluster** (Replication): When taking part in Cluster replication of tables containing `BLOB` columns, `mysqld` falsely reported a large memory leak in the replication buffers when there was none. (Bug#19247)
- **NDB Cluster**: Stopping multiple nodes could cause node failure handling not to be completed. (Bug#19039)
- **NDB Cluster**: A simultaneous `DROP TABLE` and table update operation utilising a table scan could trigger a node failure. (Bug#18597)
- **NDB Cluster**: `ndbd` could sometimes fail to start with the error `NODE FAILURE HANDLING NOT COMPLETED` following a graceful restart. (Bug#18550)
- **NDB Cluster**: Backups could fail for large clusters with many tables, where the number of tables approached `MaxNoOfTables`. (Bug#17607)
- **NDB Cluster**: A 5.1.6 or newer server did not read local checkpoints recorded by any other 5.1 version, thus preventing a system restart following an upgrade. (Bug#19333)
- **NDB Cluster**: Trying to restore the `apply_status` table from a 5.0 cluster backup failed on a 5.1 server. (Bug#18935)
- **NDB Cluster** (Disk Data): `CREATE LOGFILE GROUP` accepted values other than `NDB` or `NDBCLUSTER` in the `ENGINE` clause. (Bug#18604)
- **NDB Cluster** (Disk Data): Omitting the required `ENGINE` clause from a `CREATE LOGFILE GROUP` or `CREATE TABLESPACE` statement caused the server to crash. An appropriate error message is now returned instead. (Bug#18603)
- **NDB Cluster**: `mysqldump` included in its output data from the internal `cluster` database. (Bug#17840)
- Event-creation statements enclosed in multi-line comments using `/*!version_number ... */` syntax were not parsed correctly. (Bug#18078)
- Within a trigger, `CONNECTION_ID()` did not return the connection ID of the thread that caused the trigger to be activated. (Bug#16461)
- `mysqltest` incorrectly interpreted some `ER_XXX` error names given in the `error` command. (Bug#18495)
- For single-`SELECT` union constructs of the form `(SELECT ... ORDER BY order_list1 [LIMIT n]) ORDER BY order_list2`, the `ORDER BY` lists were concatenated and the `LIMIT` clause was ignored. (Bug#18767)
- Logging to the `mysql.general_log` and `mysql.slow_log` tables did not work for Windows builds because the `CSV` storage engine was unavailable. The `CSV` engine now is enabled in Windows builds. (Bug#17368)
- `LOAD DATA FROM MASTER` would fail when trying to load the `INFORMATION_SCHEMA` database from the master, because the `INFORMATION_SCHEMA` system database would already exist on the slave. (Bug#18607)
- The binary log would create an incorrect `DROP` query when creating temporary tables during replication. (Bug#17263)
- `CREATE VIEW` statements would not be replicated to the slave if the `--replicate-wild-ignore-table` rule was enabled.

([Bug#18715](#))

- In `mysqltest`, `--sleep=0` had no effect. Now it correctly causes `sleep` commands in test case files to sleep for 0 seconds. ([Bug#18312](#))
- Attempting to set the default value of an `ENUM` or `SET` column to `NULL` caused a server crash. ([Bug#19145](#))
- Index corruption could occur in cases when `key_cache_block_size` was not a multiple of `myisam_block_size` (for example, with `key_cache_block_size=1536` and `myisam_block_size=1024`). ([Bug#19079](#))
- The `sql_big_selects` system variable was not displayed by `SHOW VARIABLES`. ([Bug#17849](#))
- The `sql_notes` and `sql_warnings` system variables were not always displayed correctly by `SHOW VARIABLES` (for example, they were displayed as `ON` after being set to `OFF`). ([Bug#16195](#))
- `LAST_INSERT_ID()` in a stored function or trigger returned zero. . ([Bug#15728](#))
- Use of `CONVERT_TZ()` in a view definition could result in spurious syntax or access errors. ([Bug#15153](#))
- The `system_time_zone` and `version_*` system variables could not be accessed via `SELECT @@var_name` syntax. ([Bug#12792](#), [Bug#15684](#))
- Conversion of a number to a `CHAR UNICODE` string returned an invalid result. ([Bug#18691](#))
- Some fast `ALTER TABLE` operations (requiring no temporary table) did not work for all tables. ([Bug#19011](#))
- `DELETE` and `UPDATE` statements that used large `NOT IN (value_list)` clauses could use large amounts of memory. ([Bug#15872](#))
- Prevent recursive views caused by using `RENAME TABLE` on a view after creating it. ([Bug#14308](#))
- A `LOCK TABLES` statement that failed could cause `MyISAM` not to update table statistics properly, causing a subsequent `CHECK TABLE` to report table corruption. ([Bug#18544](#))
- A failed `ALTER TABLE` operation could fail to clean up a temporary `.frm` file. ([Bug#18129](#))
- For a reference to a non-existent stored function in a stored routine that had a `CONTINUE` handler, the server continued as though a useful result had been returned, possibly resulting in a server crash. ([Bug#18787](#))
- `InnoDB` did not use a consistent read for `CREATE ... SELECT` when `innodb_locks_unsafe_for_binlog` was set. ([Bug#18350](#))
- `InnoDB` could read a delete mark from its system tables incorrectly. ([Bug#19217](#))
- `myisamchk` and `myisam_ftdump` should allow either table names or `.MYI` filenames as arguments, but allowed only table names. ([Bug#19220](#))
- `DROP DATABASE` did not drop stored routines associated with the database if the database name was longer than 21 characters. ([Bug#18344](#))
- Avoid trying to include `<asm/atomic.h>` when it doesn't work in C++ code. ([Bug#13621](#))
- Executing `SELECT` on a large table that had been compressed within `myisampack` could cause a crash. ([Bug#17917](#))
- `NDB Cluster` (NDBAPI): Passing a nonexistent index name to `NdbIndexScanOperation::setBound()` caused a segmentation fault. ([Bug#19088](#))
- `ALTER TABLE ... REBUILD PARTITION` returned an inaccurate error message. ([Bug#16819](#))
- `InnoDB`: A `DELETE` followed by an `INSERT` and then by an `UPDATE` on a partitioned `InnoDB` table caused subsequent queries to return incorrect results. ([Bug#17992](#))
- `NDB Cluster`: A table insert or update of more than 128 bytes of data in a 4-replica Cluster could cause a node to crash. ([Bug#18622](#))
- `NDB Cluster` (Disk Data): Concurrent table schema operations and operations on log files groups, tablespaces, data files, or un-

dofiles could lead to Cluster node failures. (Bug#18575)

- **NDB Cluster**: An issue with replication caused a `mysqld` connected to a replicated cluster to crash when entering single user mode. (Bug#18535)
- Successive `ALTER TABLE ... DROP PARTITION` statements on the same subpartitioned table could eventually cause the server to crash. (Bug#18962)
- **NDB Cluster**: When attempting to create an index on a `BIT` or `BLOB` column, `ERROR 743: UNSUPPORTED CHARACTER SET IN TABLE OR INDEX` was returned instead of `ERROR 906: UNSUPPORTED ATTRIBUTE TYPE IN INDEX`.
- **NDB Cluster**: The Cluster binlog `mysqld` accepted updates even though the binary log was not set up, which could lead to updates missing from the binary log. (Bug#18932)
- **NDB Cluster**: Concurrent `INSERT` and `ROLLBACK` statements from different connections could cause node failures. (Bug#19245)
- **NDB Cluster** (Disk Data): Running an `INSERT` and a `DELETE` on a Disk Data table in the same transaction could cause a deadlock. (Bug#19244)
- **NDB Cluster** (Disk Data): Issuing a `CREATE LOGFILE GROUP` statement during the drop of an `NDB` table would cause database corruption. (Bug#19141)
- **NDB Cluster**: `ndb_restore` failed to restore a backup made from a 5.0 cluster to a 5.1 cluster. (Bug#18210)
- **NDB Cluster**: Adding an index to an unsigned integer column did not work correctly. (Bug#18133)
- **NDB Cluster**: A `SELECT` from an `NDB` table with `ORDER BY indexed_column` and a `LIMIT` clause would fail following `ALTER TABLE`. (Bug#18094)
- **NDB Cluster**: Performing multiple `ALTER TABLE` operations on the same `NDB` table from different `mysqld` processes in the same cluster led to schema versioning errors when trying to access the table again following the restart of one of the `mysqld` processes. (Bug#16445)
- **NDB Cluster**: Starting `mysqld` without `--log-bin` caused DDL statements on `NDB` tables to time out. (Bug#19214)
- **NDB Cluster**: Fragment IDs were not being logged correctly, causing `ndb_restore_log` to fail. (Bug#18594)
- Casting a string to `DECIMAL` worked, but casting a trimmed string (using `LTRIM()` or `RTRIM()`) resulted in loss of decimal digits. (Bug#17043)
- **NDB Cluster** (Replication): Delete and update of rows in a table without a primary key failed on the slave. (Bug#17400)
- **NDB Cluster**: On slow networks or CPUs, the management client `SHOW` command could sometimes erroneously show all data nodes as being master nodes belonging to nodegroup 0. (Bug#15530)
- The XPath `string-length()` function was not implemented for use with `ExtractValue()`. (Bug#16319)
- Fix the way that Instance Manager finds the version number of instances, so that it works properly when the executable name isn't the same as what the Instance Manager launched (such as when wrapping a `libtool`-wrapped executable from the source tree). (Bug#19059)
- The server attempted to flush uninitialized log tables during `SIGHUP` processing, causing a crash. (Bug#18848)
- If the second or third argument to `BETWEEN` was a constant expression such as `'2005-09-01' - INTERVAL 6 MONTH` and the other two arguments were columns, `BETWEEN` was evaluated incorrectly. (Bug#18618)
- If the first argument to `BETWEEN` was a `DATE` or `TIME` column of a view and the other arguments were constants, `BETWEEN` did not perform conversion of the constants to the appropriate temporary type, resulting in incorrect evaluation. (Bug#16069)
- `ExtractValue` function did not return character data within `<![CDATA[]]>` as expected. (Bug#18285)
- Server and clients ignored the `--sysconfdir` option that was passed to `configure`. (Bug#15069)
- It was possible to create a `RANGE`-partitioned table with a partition defined using the clause `VALUES LESS THAN (NULL)`, even though such a partition could never contain any values whatsoever. (Bug#18752)

- Running an `ALTER TABLE` on a partitioned table simultaneously experiencing a high number of concurrent DML statements could crash the server. (Bug#18572)
- It was possible to use trailing spaces in the names of partitions and subpartitions. Attempting to do so now raises the error `INCORRECT PARTITION NAME`. (Bug#17973)
- `LIKE` searches failed on a `CHAR` column used as the partitioning column of a table partitioned by `KEY`. (Bug#17946)
- If the `WHERE` condition of a query contained an `OR`-ed `FALSE` term, the set of tables whose rows cannot serve for null-complements in outer joins was determined incorrectly. This resulted in blocking possible conversions of outer joins into joins by the optimizer for such queries. (Bug#17164)
- The `ExtractValue()` function failed with a syntax error when the XPath expression used special characters such as `ñ` (“N-tilde”). (Bug#16233)
- Inserts failed with duplicate key errors on a table partitioned using an `AUTO_INCREMENT` column for the partitioning key. (Bug#18552, Bug#18753)
- Delimited identifiers for partitions were not being treated the same as delimited identifiers for other database objects (such as tables and columns) with regard to allowed characters. (Bug#18750)
- A query on a table partitioned or subpartitioned by `HASH` did not display all results when using a `WHERE` condition involving a column used in the hashing expression. (Bug#18423, Bug#18329)
- If the server were built without partition support, it was possible to run partitioning-related statements with no errors or warnings, even though these statements would have no effect. Now such statements are disallowed unless the server has been compiled using the `--with-partition` option. (Bug#15561)
- **NDB Cluster**: In a 2-node cluster with a node failure, restarting the node with a low value for `StartPartialTimeout` could cause the cluster to come up partitioned (“split-brain” issue). (Bug#16447)

A similar issue could occur when the cluster was first started with a sufficiently low value for this parameter. (Bug#18612)
- **NDB Cluster**: On systems with multiple network interfaces, data nodes would get “stuck” in startup phase 2 if the interface connecting them to the management server was working on node startup while the interface interconnecting the data nodes experienced a temporary outage. (Bug#15695)
- **NDB Cluster**: Unused open handlers for tables in which the metadata had changed were not properly closed. This could result in stale results from Cluster tables following an `ALTER TABLE`. (Bug#13228)
- **NDB Cluster**: Uninitialized internal variables could lead to unexpected results. (Bug#11033, Bug#11034)
- The presence of multiple equalities in a condition after reading a constant table could cause the optimizer not to use an index. This resulted in certain queries being much slower than in MySQL 4.1. (Bug#16504)
- A recent change caused the `mysql` client not to display `NULL` values correctly and to display numeric columns left-justified rather than right-justified. The problems have been corrected. (Bug#18265)
- **InnoDB** failure to release an adaptive hash index latch could cause a server crash if the query cache was enabled. (Bug#15758)
- **InnoDB**: `ALTER TABLE` to add or drop a foreign key for an **InnoDB** table had no effect. (Bug#18477)
- **NDB Cluster**: Attempting to create an index using multiple columns on an explicitly partitioned table in a replicated Cluster database could cause the master `mysqld` process to crash. (Bug#18284)
- **NDB Cluster**: Queries using `ORDER BY pkN` failed against a `LIST`-partitioned Cluster table having a multi-column primary key, where `pkN` represents one of the columns making up the primary key. (Bug#18598)
- Updating a field value when also requesting a lock with `GET_LOCK()` would cause slave servers in a replication environment to terminate. (Bug#17284)

C.1.21. Changes in release 5.1.9 (12 April 2006)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on *production*-level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details please see <http://www.mysql.com/products/enterprise>.

Functionality added or changed:

- `SHOW PLUGIN` was renamed to `SHOW PLUGINS`. `SHOW PLUGIN` now is deprecated and generates a warning. (Bug#17112)
- Binary MySQL distributions now include a `mysqld-max` server, in addition to the usual `mysqld` optimized server and the `mysqld-debug` debugging server.
- `mysqld_safe` no longer checks for a `mysqld-max` binary. Instead, `mysqld_safe` nows checks only for the standard `mysqld` server unless another server binary is specified explicitly via `--mysqld` or `--mysqld-version`. If you previously relied on the implicit invocation of `mysqld-max`, you should use an appropriate option now. (Bug#17861)
- For partitioned tables, the output of `SHOW TABLE STATUS` now shows in the `Engine` column the name of the storage engine used by all partitions for the table; in the `Create_options` column, the output now shows `partitioned` for a partitioned table. This change also affects the values shown in the corresponding columns of the `INFORMATION_SCHEMA.TABLES` table. (Bug#17631)
- **NDB Cluster:** A new `--nowait-nodes` startup option for `ndbd` makes it possible to “skip” specific nodes without waiting for them to start when starting the cluster. See [Section 16.6.5.2, “Command Options for ndbd”](#).
- The **NDBCluster** storage engine now supports `CREATE TABLE` statements of arbitrary length. (Previously, `CREATE TABLE` statements for MySQL Cluster tables could contain a maximum of 4096 characters only.) (Bug#17813)
- Large file support was re-enabled for the MySQL server binary for the AIX 5.2 platform. (Bug#13571)
- The `mysql_get_ssl_cipher()` C API function was added.

Bugs fixed:

- **Security fix:** Invalid arguments to `DATE_FORMAT()` caused a server crash. (CVE-2006-3469, Bug#20729) Thanks to Jean-David Maillefer for discovering and reporting this problem to the Debian project and to Christian Hammers from the Debian Team for notifying us of it.
- **NDB Cluster:** `BLOB` columns did not work correctly with user-partitioned **NDB** tables. (Bug#16796)
- `mysql_config` returned incorrect libraries on `x86_64` systems. (Bug#13158)
- `mysql_reconnect()` sent a `SET NAMES` statement to the server, even for pre-4.1 servers that do not understand the statement. (Bug#18830)
- `COUNT(*)` on a **MyISAM** table could return different results for the base table and a view on the base table. (Bug#18237)
- For `mysql.server`, if the `basedir` option was specified after `datadir` in an option file, the setting for `datadir` was ignored and assumed to be located under `basedir`. (Bug#16240)
- For full-text searches in boolean mode, and when a full-text parser plugin was used, a `MYSQL_FTPARSER_PARAM::ftparser_state` could have been corrupted by recursive calls to the plugin. (Bug#18836)
- `EXTRACT(QUARTER FROM date)` returned unexpected results. (Bug#18100)
- `TRUNCATE` did not reset the `AUTO_INCREMENT` counter for **MyISAM** tables when issued inside a stored procedure. (Bug#14945)

■ Note

This bug did not affect **InnoDB** tables. Also, **TRUNCATE** does not reset the **AUTO_INCREMENT** counter for **NDB-Cluster** tables regardless of when it is called (see [Bug#18864](#)).

- The server was always built as though **--with-extra-charsets=complex** had been specified. ([Bug#12076](#))
- Partition pruning did not work properly for some kinds of partitioning and subpartitioning, with certain **WHERE** clauses. (Partitions and subpartitions that should have been marked as used were not so marked.) The error could manifest as incorrect content in **EXPLAIN PARTITIONS** output as well as missing rows in the results of affected queries. ([Bug#18558](#))
- **NDB Cluster**: An uninitialized internal variable could lead to unexpected results. ([Bug#18831](#))
- For tables created in a MySQL 4.1 installation upgraded to MySQL 5.0 and up, multiple-table updates could update only the first matching row. ([Bug#16281](#))
- Complex queries with nested joins could cause a server crash. ([Bug#18279](#))
- A query against a partitioned table using **WHERE col IS NULL** could produce incorrect results given the following conditions:
 - The table had partitions and subpartitions
 - The partitioning function depended on a single column **col** of one of the MySQL integer types
 - The partitioning function was not monotonically increasing

The same issue could cause the server to crash when run in debug mode. ([Bug#18659](#))

- **CAST(double AS SIGNED INT)** for large **double** values outside the signed integer range truncates the result to be within range, but the result sometimes had the wrong sign, and no warning was generated. ([Bug#15098](#))
- **MEDIUMINT** columns were not handled in the same way as other column types by partition pruning.

Partition pruning would sometimes use inappropriate columns in performing queries.

Both of these issues were rectified as part of the same bugfix. ([Bug#18025](#))

- Quoted values could not be used for partition option values. ([Bug#13520](#))
- Delimited identifiers could not be used in defining partitions. ([Bug#13433](#))
- Building the server using **--with-example-storage-engine** failed to enable the **EXAMPLE** storage engine in the server. ([Bug#18464](#))
- Triggers created in one version of the server could not be dropped after upgrading to a newer version. ([Bug#15921](#))
- Queries using **WHERE ... IS NULL** returned incorrect results from partitioned tables. ([Bug#18070](#))
- Partition pruning did not perform correctly with partitions on **NULL**, and could potentially crash the server. ([Bug#18053](#))
- If **InnoDB** encountered a **HA_ERR_LOCK_TABLE_FULL** error and rolled back a transaction, the transaction was still written to the binary log. ([Bug#18283](#))

C.1.22. Changes in release 5.1.8 (Not released)

Note

This was an internal release only, and no binaries were published.

Functionality added or changed:

- In order not to break legacy applications, support for **TYPE = engine_name** has been restored, but now generates a warning.

Important

This option has been deprecated since MySQL 4.0. Beginning with MySQL 5.2, `TYPE = engine_name` will no longer be available and will produce a syntax error. You should not use `TYPE` in any new applications, and you should immediately begin conversion of existing applications to use the `ENGINE = engine_name` syntax instead. (Bug#17501)

- The deprecated constructs in the following table now generate warnings. You should not employ them in new applications, as they are likely to be removed in a future version of MySQL. Use the equivalents shown in the table's second column instead. For the same reason, existing applications depending on the deprecated constructs should be converted to make use of the current equivalents as soon as possible. (Bug#17501)

Deprecated / Obsolete:	Current / Preferred:
<code>@@table_type</code>	<code>@@storage_engine</code>
<code>@@log_bin_trust_function_creators</code>	<code>@@log_bin_trust_function_creators</code>
<code>TIMESTAMP(N)</code>	See Section 11.6, “Date and Time Functions”.
<code>TYPE=</code>	<code>ENGINE=</code>
<code>BACKUP TABLE</code>	<code>mysql-dump</code> , <code>mysql-hotcopy</code> , or MySQL Administrator
<code>RESTORE TABLE</code> , <code>LOAD TABLE FROM MASTER</code>	<code>mysql-dump</code> , <code>mysql</code> , or MySQL Administrator
<code>SHOW TABLE TYPES</code>	<code>SHOW [STORAGE] ENGINES</code>
<code>SHOW INNODB STATUS</code>	<code>SHOW ENGINE INNODB STATUS</code>
<code>SHOW MU-TEXT STATUS</code>	<code>SHOW ENGINE INNODB MU-TEXT</code>
<code>SHOW BDB LOGS</code> , <code>SHOW LOGS</code>	<code>SHOW ENGINE BDB LOGS</code>

The deprecated items shown in the table are not guaranteed to be available in MySQL 5.2 or later.

- Incompatible change:** For purposes of determining placement, `RANGE` partitioning now treats `NULL` as less than any other value. (Formerly, `NULL` was treated as equal to zero.) See Section 17.2.6, “How MySQL Partitioning Handles `NULL` Values”. (Bug#15447)

- **Incompatible Change:** The semantics of `ALTER TABLE t ENGINE=X;` for partitioned tables is changed, and now means that the storage engine used for table *t* is changed to *X*.

The previous statement formerly (prior to MySQL 5.1.8) meant that all partitioning was removed from the table. In order to remove the partitioning of a table, the syntax `ALTER TABLE t REMOVE PARTITIONING;` is introduced. The `REMOVE PARTITIONING` option can be used in combination with existing `ALTER TABLE` options such as those employed for adding or dropping columns or indexes. (Bug#17754)

- Added the `--sysdate-is-now` option to `mysqld` to enable `SYSDATE()` to be treated as an alias for `NOW()`. See [Section 11.6, “Date and Time Functions”](#). (Bug#15101)
- The `NDBCluster` storage engine now supports `INSERT IGNORE` and `REPLACE` statements. Previously, these statements failed with an error. (Bug#17431)
- Events no longer support times past the end of the Unix epoch. (Formerly, such dates were interpreted as being at the beginning of the Unix epoch.) (Bug#16396)
- Event names are now case-insensitive. That is (for example), you cannot have events with the names `Myevent` and `MyEvent` belonging to the same database and definer. (Bug#16415)
- Builds for Windows, Linux, and Unix (except AIX) platforms now have SSL support enabled, in the server as well as in the client libraries. Because part of the SSL code is written in C++, this does introduce dependencies on the system's C++ runtime libraries in several cases, depending on compiler specifics. (Bug#18195)
- Temporary tables may no longer be partitioned. (Bug#17497)
- Added the `--events` option to `mysqldump` to enable events to be included in the dump output. (Bug#16853)
- **NDB Cluster (Disk Data):** You can now have only one log file group at any one time. See [Section 12.1.8, “CREATE LOGFILE GROUP Syntax”](#). (Bug#16386)
- The syntax for `CREATE PROCEDURE` and `CREATE FUNCTION` statements now includes a `DEFINER` clause. The `DEFINER` value specifies the security context to be used when checking access privileges at routine invocation time if the routine has the `SQL SECURITY DEFINER` characteristic. See [Section 19.2.1, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#), for more information.

When `mysqldump` is invoked with the `--routines` option, it now dumps the `DEFINER` value for stored routines.

- The output from `SHOW CREATE TABLE` is more consistent about using uppercase for keywords. Data types still are in lowercase. (Bug#10460)
- The `ExtractValue()` function with `contains()` now uses the SQL collation in making comparisons. Previously, comparisons were always binary (that is, case-sensitive). (Bug#16316)
- The `cluster_replication` database has been renamed to `cluster`. This will effect replication between MySQL Clusters where one cluster is running MySQL 5.1.8 or later, and the other is running MySQL 5.1.7 or earlier. See [Section 16.10, “MySQL Cluster Replication”](#), and especially [Section 16.10.4, “Cluster Replication Schema and Tables”](#).
- The stability of `CREATE` and `DROP` operations on `NDB` tables containing `BLOB` columns has been improved. (Bug#17761)
- More specific error messages are now given when attempting to create an excessive number of partitions or subpartitions. (Previously, no distinction was made between an excessive number of partitions and an excessive number of subpartitions.) (Bug#17393)
- The `mysqltest` utility now converts all `CR/LF` combinations to `LF` to allow test cases intended for Windows to work properly on UNIX-like systems. (Bug#13809)
- The `mysql_ping` function will now retry if the `reconnect` flag is set and error `CR_SERVER_LOST` is encountered during the first attempt to ping the server. (Bug#14057)
- `mysqldump` now surrounds the `DEFINER`, `SQL SECURITY DEFINER` and `WITH CHECK OPTION` clauses of a `CREATE VIEW` statement with "not in version" comments to prevent errors in earlier versions of MySQL. (Bug#14871)
- For an event having no `STARTS` time specified when it was created, the `mysql.event` table's `start` column now displays the creation time rather than `NULL`. (Bug#16537)

In addition, both the `SHOW EVENTS` statement's `Starts` column and the `STARTS` column of the `INFORMATION_SCHEMA.EVENTS` table are now empty rather than `NULL` when `STARTS` was not used in the `CREATE EVENT` statement.

- `MICROSECOND` intervals are no longer allowed for events. (Bug#16411)
- Description of the `EVENT` privilege has been changed to `To create, alter, drop, and execute events`. (Bug#16412)
- The `binlog_format` system variable now is dynamic and can be changed at runtime, as described in Section 15.1.2, “Replication Formats”.
- The `binlog_format` system variable now can be set to a third format, `MIXED`, as described in Section 15.1.2, “Replication Formats”.
- A slave server may switch the format *automatically* now. This happens when the server is running in either `STATEMENT` or `MIXED` format and encounters a row in the binary log that is written in `ROW` logging format. In that case, the slave switches to row-based replication temporarily for that event, and switches back to the previous format afterwards.
- Partition pruning was made more stable, particularly in cases involving queries using tests for `NULL` values in the `WHERE` clause against subpartitioned tables which were partitioned by `LIST( some_function(col1, ... ,colN) )`. (Bug#17891)
- Names of subpartitions must now be unique for an entire table, and not merely within the same partition. (Bug#15408)
- The output of `SHOW CREATE EVENT` no longer qualifies the event name with the name of the schem to which the event belongs. (Bug#17714)
- The client API will now attempt to reconnect on TCP/IP if the `reconnect` flag is set, as is the case with sockets. (Bug#2845)
- The XPath `last()` function is now implemented for use with `ExtractValue()`. (Bug#16318)

Bugs fixed:

- Stored routine names longer than 64 characters were silently truncated. Now the limit is properly enforced and an error occurs. (Bug#17015)
- During conversion from one character set to `ucs2`, multi-byte characters with no `ucs2` equivalent were converted to multiple characters, rather than to `0x003F QUESTION MARK`. (Bug#15375)
- Slave servers would retry the execution of a SQL statement an infinite number of times, ignoring the value `SLAVE_TRANSACTION_RETRIES` when using the NDB engine. (Bug#16228)
- Replication of data stored in a partitioned table would cause slave servers to issue a assertion and terminate. (Bug#18436)
- The `mysql_close()` C API function leaked handles for shared-memory connections on Windows. (Bug#15846)
- Checks for permissions on database operations could be performed in a case-insensitive manner (a user with permissions on database `MYDATABASE` could by accident get permissions on database `myDataBase`), if the privilege data were still cached from a previous check. (Bug#17279)
- The server would crash when `SHOW STATUS` was called on a server linked with `yaSSL`. (Bug#18310)
- `SELECT ... WHERE column LIKE 'A%'`, when `column` had a key and used the `latin2_czech_cs` collation, caused the wrong number of rows to be returned. (Bug#17374)
- Using `ALTER TABLE` to increase the length of a `BINARY(M)` column caused column values to be padded with spaces rather than `0x00` bytes. (Bug#16857)
- Execution of a stored function or trigger which inserted data into a table while running concurrent selects on the same table could result in storing incorrect data in the query cache. (Bug#14767)
- **NDB Cluster**: Attempting to restart a node with dropped events still pending would fail. (Bug#18491)
- **NDB Cluster**: In asynchronous replication scenarios, binary log events could be lost on the remote `mysqld`. (Bug#18472)

- **NDB Cluster**: Two `mysqld` processes starting at the same time could cause a race condition. (Bug#18472)
- **NDB Cluster**: Two `mysqld` processes did not synchronise `DROP TABLE` binary log events correctly. (Bug#18395)
- **NDB Cluster**: When multiple node restarts were attempted without allowing each restart to complete, the error message returned was `ARRAY INDEX OUT OF BOUNDS` rather than `TOO MANY CRASHED REPLICAS`. (Bug#18349)
- A `SELECT ... ORDER BY ...` from a view defined using a function could crash the server. An example of such a view might be `CREATE VIEW v1 AS SELECT SQRT(c1) FROM t1`. (Bug#18386)
- `REPAIR TABLE`, `OPTIMIZE TABLE`, and `ALTER TABLE` operations on transactional tables (or on tables of any type on Windows) could corrupt triggers associated with those tables. (Bug#18153)
- **MyISAM**: Performing a bulk insert on a table referenced by a trigger would crash the table. (Bug#17764)
- Using `ORDER BY intvar` within a stored procedure (where `intvar` is an integer variable or expression) would crash the server. (Bug#16474)

Note

The use of an integer `i` in an `ORDER BY i` clause for sorting the result by the `ith` column is deprecated (and non-standard). It should *not* be used in new applications. See [Section 12.2.7, “SELECT Syntax”](#).

- A `SELECT` using a function against a nested view would crash the server. (Bug#15683)
- `ALTER TABLE ... ADD COLUMN ... AFTER ...` failed when used on partitioned tables. (Bug#16806)
- **NDB Cluster**: A timeout in the handling of an `ABORT` condition with more than 32 operations could yield a node failure. (Bug#18414)
- **NDB Cluster**: A node restart immediately following a `CREATE TABLE` would fail. **Important**: This fix supports 2-node Clusters only. (Bug#18385)
- **NDB Cluster**: In event of a node failure during a rollback, a “false” lock could be established on the backup for that node, which lock could not be removed without restarting the node. (Bug#18352)
- **NDB Cluster**: The cluster created a crashed replica of a table having an ordered index — or when logging was not enabled, of a table having a table or unique index — leading to a crash of the cluster following 8 successive restarts. (Bug#18298)
- **NDB Cluster**: When replacing a failed master node, the replacement node could cause the cluster to crash from a buffer overflow if it had an excessively large amount of data to write to the cluster log. (Bug#18118)
- **NDB Cluster**: Restarting nodes were allowed to start and join the cluster too early. (Bug#16772)
- **NDB Cluster**: Issuing a `DROP LOGFILE GROUP` statement would cause `ndbd` processes to crash if MySQL had been compiled with `gcc4`. (Bug#18295)
- Using triggers with partitioned **InnoDB** tables led to incorrect results. (Bug#17744)
- Calling `CREATE TABLE` or `ALTER TABLE` twice on a partitioned table in a stored procedure or a prepared statement resulted in errors and sometimes server crashes. (Bug#17290)
- A problem with `NULL`s and interval mapping sometimes caused incorrect results or crashes when trying to use less-than searches on partitioned tables. (Bug#17173)
- `CREATE TABLE ... PARTITION ... AS SELECT ...` would cause the server to crash. (Bug#15336)
- Creating a partition which depends on an expression containing a column using the UTF8 character set would cause the server to crash. (Bug#14367)
- Invoking more than once a prepared statement that creates a partitioned table would crash the server. (Bug#14350)
- **NDB Cluster**: A `SELECT ... ORDER BY` query on an explicitly partitioned Cluster table with no explicit indexes would crash the server. (Bug#17899)
- The `ExtractValue()` function did not return an error when passed an invalid XPath string. (Bug#18172)

- Stored procedures that call UDFs and pass local string variables caused server crashes. ([Bug#17261](#))
- Connecting to a server with a UCS2 default character set with a client using a non-UCS2 character set crashed the server. ([Bug#18004](#))
- Loading of UDFs in a statically linked MySQL caused a server crash. UDF loading is now blocked if the MySQL server is statically linked. ([Bug#11835](#))
- A security enhancement in Visual Studio 8 could cause a MySQL debug server compiled with it to hang when running `SELECT` queries against partitioned tables. ([Bug#17722](#))
- **NDB Cluster**: `auto_increment` values were not propagated correctly in statement-based replication. ([Bug#18208](#))
- Repeated invocations of a stored procedure containing a `SHOW CREATE EVENT` statement would result in the error `PACKETS OUT OF ORDER`. ([Bug#17403](#))
- Repeated invocations of a stored procedure containing a `CREATE EVENT` or `ALTER EVENT` statement would crash the server. ([Bug#16408](#))
- Renaming and adding a new column to a partitioned table in the same `ALTER TABLE` statement caused the server to crash. ([Bug#17772](#))
- `ALTER TABLE ... REBUILD PARTITION` with no partition name specified would crash the server. ([Bug#17940](#))
- Trying to add a partition to a table having subpartitions could crash the server. ([Bug#17140](#))
- Attempting to use a conflicting `VALUES` clause in `ALTER TABLE ... ADD PARTITION` caused the server to crash. An example of such a conflicting clause would be that uses `VALUES LESS THAN (constant)` (which indicates a range) with a table that is partitioned by `LIST`. ([Bug#17127](#))
- `ALTER TABLE ... COALESCE PARTITION` failed with an `OUT OF MEMORY` error. ([Bug#16810](#))
- Names of subpartitions were not displayed in the output of `SHOW CREATE TABLE`. ([Bug#16370](#))
- Setting up subpartitions on at least one but not all the partitions of a partitioned table caused the server to crash. ([Bug#15407](#))
- Using the `position()` function in the XPath argument to `ExtractValue()` crashed the server. ([Bug#18171](#))
- A query with a `WHERE date_column > date_value` condition failed on a table partitioned by `RANGE`. ([Bug#17894](#))
- Cursors in stored routines could cause a server crash. ([Bug#16887](#))
- Replication slaves could not replicate triggers from older servers that included no `DEFINER` clause in the trigger definition. Now the trigger executes with the privileges of the invoker (which on the slave is the slave SQL thread). ([Bug#16266](#))
- Character set conversion of string constants for `UNION` of constant and table column was not done when it was safe to do so. ([Bug#15949](#))
- `NULL` values were written to the `mysql.slow_log` table incorrectly. ([Bug#17600](#))
- A query with a `WHERE date_column > date_value` condition failed on a table partitioned by `RANGE`. ([Bug#17894](#))
- A failed `ALTER TABLE ... ADD PRIMARY KEY` on a partitioned table would result in bad table metadata and could possibly crash the server. ([Bug#17097](#))
- No error was reported when subpartitions were defined for a non-subpartitioned table. ([Bug#15961](#))
- Searches on indexed columns of partitioned tables failed to find all matching rows following updates of the indexed columns. ([Bug#14526](#))
- The `DEFINER` value for stored routines was not replicated. ([Bug#15963](#))
- Use of `TRUNCATE TABLE` for a `TEMPORARY` table on a master server was propagated to slaves properly, but slaves did not decrement the `Slave_open_temp_tables` counter properly. ([Bug#17137](#))
- `SELECT COUNT(*)` for a `MyISAM` table could return different results depending on whether an index was used. ([Bug#14980](#))

- Updating a view that filters certain rows to set a filtered out row to be included in the table caused infinite loop. For example, if the view has a WHERE clause of `salary > 100` then issuing an UPDATE statement of `SET salary = 200 WHERE id = 10`, caused an infinite loop. (Bug#17726)
- Creating a table with the same name as the mapped name of another table caused a server crash. For example, if MySQL maps the table name `txu#P#p1` to `txu@0023P@0023p1` on disk, creating another table named `txu@0023P@0023p1` crashed the server. (Bug#17142)
- **NDB Cluster**: Adding an index together with replication could cause `mysqld` to crash. (Bug#18106)
- **NDB Cluster**: Insufficient `StringBuffer` memory when attempting to create a trigger caused the server to crash. (Bug#18101)
- **NDB Cluster**: Variable-length columns used as primary keys were not handled correctly. (Bug#18075)
- **NDB Cluster**: Row-based replication could fail with tables using `VARCHAR` columns for primary keys and having `BLOB` columns. (Bug#18067)
- **NDB Cluster**: `CREATE UNIQUE INDEX` on a column containing non-unique data could cause one or more `ndbd` nodes to hang or crash. (Bug#18040)
- **NDB Cluster** (Disk Data): `CREATE UNIQUE INDEX` failed with `ERROR 4243: INDEX NOT FOUND`. (Bug#18039)
- **NDB Cluster**: Node recovery of tables with `VARCHAR` columns using character sets was inconsistent, which could cause a number of issues, including the data nodes failing to restart and `ALTER TABLE` statements to hang. (Bug#18026)
- **NDB Cluster**: In some cases, a single `ndbd` node would fail following a system restart. (Bug#17854)
- **NDB Cluster** (Replication): The binary log on the secondary master was not being set up correctly following a table rename. (Bug#17838)
- **NDB Cluster**: With a single replica, transactions waiting in the log synchronisation queue were not being restarted, causing them to be aborted. (Bug#17536)
- **NDB Cluster** (Disk Data): It was not possible to create more than 9 tablespaces. (Bug#16913)
- **NDB Cluster**: Inserting and deleting `BLOB` column values while a backup was in process could cause the loss of an `ndbd` node. (Bug#14028)
- **NDB Cluster**: `UNDO_BUFFER_SIZE` was limited to 17 MB. (Bug#16657, Bug#17890)
- Using `ALTER TABLE ... REBUILD PARTITION` without specifying the name of the partition caused the server to crash, rather than reporting a syntax error. (Bug#17947)
- When attempting to insert a `0` into a `LIST`-partitioned table that had no value-list containing `0`, no error was reported. (Bug#15253)
- If the server was started with the `--skip-grant-tables` option, it was impossible to create a trigger or a view without explicitly specifying a `DEFINER` clause. (Bug#16777)
- The server would execute stored routines that had a non-existent definer. (Bug#13198)
- **NDB Cluster**: A simultaneous `RENAME` of several tables was logged multiple times. (Bug#17827)
- **NDB Cluster**: Trying to perform a `DELETE` from a Cluster table following a `LOCK TABLES` would cause the `ndbd` processes to hang. (Bug#17812)
- `ALTER TABLE ... REORGANIZE PARTITION` failed with `ERROR ON RENAME OF FILENAME ...` on Windows. (Bug#17720)
- **NDB Cluster**: `ALTER TABLE ... ADD INDEX` failed with `ERROR 756: INDEX ON DISK COLUMN IS NOT SUPPORTED` when run against a Disk Data table having a primary key. (Bug#17888)
- **NDB Cluster**: `DELETE` operations on `NDB` tables could cause memory leaks. (Bug#16874)
- **NDB Cluster**: Some query cache statistics were not always correctly reported for Cluster tables. (Bug#16795)

- The `EXAMPLE` storage engine did not work on Windows. ([Bug#17721](#))
- For `FEDERATED` tables, a `SELECT` statement with an `ORDER BY` clause did not return rows in the proper order. ([Bug#17377](#))
- Setting the `myisam_repair_threads` system variable to a value larger than 1 could cause corruption of large `MyISAM` tables. ([Bug#11527](#))
- The length of a `VARCHAR()` column that used the `utf8` character set would increase each time the table was re-created in a stored procedure or prepared statement, eventually causing the `CREATE TABLE` statement to fail. ([Bug#13134](#))
- The MySQL server could crash with out of memory errors when performing aggregate functions on a `DECIMAL` column. ([Bug#17602](#))
- `INSERT` statements executed by scheduled events were not written to the general log. ([Bug#16413](#))
- A memory leak caused warnings on slaves for certain statements that executed without warning on the master. ([Bug#16175](#))
- Naming a partition using the characters `Ç` or `ç` (“c-cedilla”; Unicode `00C7` or `00E7`) made unreadable the table containing the partition. ([Bug#14527](#))
- The `self()` XPath function was not handled correctly by `ExtractValue()`. ([Bug#16315](#))
- The `ExtractValue()` function would not accept expressions which matched element names containing an underscore character. ([Bug#16320](#))
- `NDB Cluster`: Trying to update very large partitioned tables using the `NDB` storage engine sometimes caused the server to crash. ([Bug#16385](#), [Bug#17806](#))
- Slow queries executed by scheduled events were not being written to the slow query log. ([Bug#16426](#))
- On Linux, creation of table partitions failed within a stored procedure. ([Bug#14363](#))
- `mysql_fix_privilege_tables` didn't create the `mysql.plugin` table. ([Bug#17568](#))
- Improper checking of binary log statements could result in a server crash. ([Bug#17457](#))
- The `ExtractValue()` function allowed the use of the `!` character in identifiers by ignoring the illegal character. This is now correctly reported as a syntax error. ([Bug#16313](#))
- `NDB Cluster`: Trying to insert a value into a nonexistent `LIST` partition of an `NDB` table would cause the server to crash. ([Bug#17763](#))
- `NDB Cluster`: `ALTER TABLE` on a partitioned `NDB` table could cause the server to crash. ([Bug#17499](#))
- `NDB Cluster`: A repeated `SELECT` on a partitioned table that used the `NDB` storage engine could cause the server to crash. ([Bug#17390](#))
- `NDB Cluster`: Using `ALTER TABLE ... ADD PARTITION` on a table partitioned by `LIST` would cause the client to hang. ([Bug#17701](#))
- Triggers created without `BEGIN` and `END` clauses resulted in “You have an error in your SQL syntax” errors when dumping and re-playing a binary log. ([Bug#16878](#))
- The `RENAME TABLE` statement did not move triggers to the new table. ([Bug#13525](#))
- Clients compiled from source with the `--without-readline` did not save command history from session to session. ([Bug#16557](#))
- Stored routines that contained only a single statement were not written properly to the dumpfile when using `mysqldump`. ([Bug#14857](#))
- Issuing `GRANT EXECUTE` on a procedure would display any warnings related to the creation of the procedure. ([Bug#7787](#))
- Attempting to add a new partition to a table partitioned by a unique key would cause an `OUT OF MEMORY` error. ([Bug#17169](#))
- In a highly concurrent environment, a server crash or deadlock could result from execution of a statement that used stored functions

or activated triggers coincident with alteration of the tables used by these functions or triggers. ([Bug#16593](#))

C.1.23. Changes in release 5.1.7 (27 February 2006)

Functionality added or changed:

- **Incompatible change:** `TYPE = engine_name` is no longer accepted as a synonym for the `ENGINE = engine_name` table option. (`TYPE` has been deprecated since MySQL 4.0.)
- Several changes were made to make upgrades easier:
 - Added the `mysql_upgrade` program that checks all tables for incompatibilities with the current version of MySQL Server and repairs them if necessary. This program should be run for each MySQL upgrade (rather than `mysql_fix_privilege_tables`). See [Section 5.5.7, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).
 - Added the `FOR UPGRADE` option for the `CHECK TABLE` statement. This option checks whether tables are incompatible with the current version of MySQL Server.
 - Added the `--check-upgrade` to `mysqlcheck` that invokes `CHECK TABLE` with the `FOR UPGRADE` option. Added the `--fix-db-names` and `--fix-table-names` options to `mysqlcheck`.
- **Incompatible change:** The `mysql_stmt_attr_get()` C API function now returns a boolean rather than an unsigned int for `STMT_ATTR_UPDATE_MAX_LENGTH`. ([Bug#16144](#))
- Added the `--wait-timeout` option to `mysqlmanager` to allow configuration of the timeout for dropping an inactive connection, and increased the default timeout from 30 seconds to 28,800 seconds (8 hours). ([Bug#12674](#), [Bug#15980](#))
- All subpartitions within a given partitioned table are now guaranteed to have unique names. ([Bug#15408](#))
- Added the `RENAME DATABASE` statement.
- The SQL mode in effect at the time an event is created or altered is recorded and used during event execution. ([Bug#16407](#))
- Added the `PROCESSLIST` table to `INFORMATION_SCHEMA`.
- Attempting to read pre-5.1.6 partitioned tables with a MySQL 5.1.7 (or later) server now generates a suitable warning message. ([Bug#16695](#))

For additional information about this issue, see [Section C.1.24, “Changes in release 5.1.6 \(01 February 2006\)”](#).
- **NDB Cluster:** Attempting to `SELECT ... FROM INFORMATION_SCHEMA.FILES` now raises a warning in the event that the cluster has crashed. ([Bug#17087](#))
- **NDB Cluster:** It is now possible to replicate NDB tables having no explicit primary key. See [Section 16.10, “MySQL Cluster Replication”](#).
- Removed the `have_isam` and `have_raid` system variables.
- Status messages added to `ndb_restore` to allow users to know that data files for Disk Data are being created. ([Bug#16873](#))
- Added the `IN NATURAL LANGUAGE MODE` and `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` modifiers for full-text searches. See [Section 11.8, “Full-Text Search Functions”](#).
- Creator privileges are now checked for all events before execution. ([Bug#17289](#))
- `CREATE EVENT`, `DROP EVENT`, and `ALTER EVENT` statements are not allowed in triggers. ([Bug#16410](#))
- New `charset` command added to `mysql` command-line client. By typing `charset name` or `\C name` (such as `\C UTF8`), the client character set can be changed without reconnecting. ([Bug#16217](#))
- In row-based replication, when executing a `Rows_log_event`, the associated table was locked, the rows applied and the lock released. This did not work since there are storage engines that count locks and perform an autocommit when the number of locks reach zero. Now we ensure that all table maps come before all `ROWS` events in a statement.

Bugs fixed:

- **NDB Cluster**: `ndbd` restart could sometimes fail due to incorrect memory access. ([Bug#17417](#))
- Execution times for scheduled events were not calculated correctly: the last execution time was used as a base rather than the actual start time. ([Bug#17494](#))
- Using an XPath expression containing `=` with `ExtractValue()` caused the server to crash. ([Bug#16242](#))
- **NDB Cluster**: An unhandled resources issue could cause node failure with a `DELETE FROM TABLE` affecting thousands of rows. ([Bug#16492](#))
- **NDB Cluster**: Cluster tables not having an explicit primary key could not be replicated. ([Bug#14541](#))
- For a transaction that used **MyISAM** and **InnoDB** tables, interruption of the transaction due to a dropped connection on a master server caused slaves to lose synchrony. ([Bug#16559](#))
- `SELECT` with `GROUP BY` on a view could cause a server crash. ([Bug#16382](#))
- `SET TRANSACTION ISOLATION LEVEL` acted like `SET SESSION TRANSACTION ISOLATION LEVEL`. That is, it set the isolation level for longer than the next transaction. ([Bug#7955](#))
- `SHOW CREATE TABLE` produced extraneous spaces after the `PRIMARY KEY` keywords. ([Bug#13883](#))
- If the query optimizer transformed a `GROUP BY` clause in a subquery, it did not also transform the `HAVING` clause if there was one, producing incorrect results. ([Bug#16603](#))
- `SUBSTRING_INDEX()` could yield inconsistent results when applied with the same arguments to consecutive rows in a query. ([Bug#14676](#))
- `myisam_ftdump` did not work for `FULLTEXT` indexes associated with a parser plugin. ([Bug#17116](#))
- `BIT` fields were not properly handled when using row-based replication. ([Bug#13418](#))
- Column counts were encoded incorrectly in the binary log for row-based logging format. ([Bug#17678](#))
- Data truncations on non-`UNIQUE` indexes could crash **InnoDB** when using multi-byte character sets. ([Bug#17530](#))
- An `ALTER DATABASE` statement on a replication master crashed the slaves. ([Bug#17521](#))
- Partitioning with certain `SUBPARTITION BY HASH` clauses caused an error when querying for a partitioned column using an `IS NULL` comparison. ([Bug#17430](#), [Bug#17432](#))
- The `mysql_fix_privilege_tables.sql` script did not properly initialize the `Event_priv` column to 'Y' for those accounts that should have the `EVENT` privilege. ([Bug#16400](#))
- **NDB Cluster**: Inserting the output of `REPEAT('some_string', some_int)` into a `BLOB` column resulted in the error `INVALID BLOB ATTRIBUTES OR INVALID BLOB PARTS TABLE`. ([Bug#17505](#))
- **NDB Cluster**: Row-based replication was not being set up correctly if a backup was already in progress. For example, connecting a `mysqld` instance to a cluster which was being backed up would result in the message `NDB: SKIPPING SETUP TABLE TEST.T1` being written to the error log. ([Bug#17459](#))
- **NDB Cluster**: `CREATE TEMPORARY TABLE` of a Cluster table would fail with an `UNSUPPORTED` error or crash the server. ([Bug#17210](#), [Bug#16552](#))
- **NDB Cluster**: `UNIQUE` keys in Cluster tables were limited to 225 bytes in length. ([Bug#15918](#))
- **NDB Cluster**: `REPLACE` failed when attempting to update a primary key value in a Cluster table. ([Bug#14007](#))
- **NDB Cluster**: Creating **NDB** tables containing `BLOB` columns but no primary key caused unpredictable behavior. ([Bug#17559](#))
- Creating an event and using a whitespace character other than space following the `DO` keyword caused a server crash. ([Bug#17453](#))
- Previously, a stored function invocation was written to the binary log as `DO func_name()` if the invocation changes data and oc-

curs within a non-logged statement, or if the function invokes a stored procedure that produces an error. These invocations now are logged as `SELECT func_name()` instead for better control over error code checking (slave servers could stop due to detecting a different error than occurred on the master). (Bug#14769)

- `CHECKSUM TABLE` returned different values on MyISAM table depending on whether the `QUICK` or `EXTENDED` options were used. (Bug#8841)
- Querying the `INFORMATION_SCHEMA.PARTITIONS` table on a non-max server caused a server crash. This also happened following the creation of a table with a very large number (hundreds) of partitions. (Bug#16591, Bug#17141)
- Attempting to add a new partition to a table partitioned by a unique key would cause an `OUT OF MEMORY` error. (Bug#17169)
- MySQL server dropped client connection for certain `SELECT` statements against views defined that used `MERGE` algorithm. (Bug#16260)
- A statement containing `GROUP BY` and `HAVING` clauses could return incorrect results when the `HAVING` clause contained logic that returned `FALSE` for every row. (Bug#14927)
- Using `GROUP BY` on column used in `WHERE` clause could cause empty set to be returned. (Bug#16203)
- `DROP DATABASE` did not drop events for the database. (Bug#16406)
- Race conditions between event creation, dropping, and execution could result in a server crash or hang. (Bug#17373)
- `SET sql_mode = N`, where $N > 31$, did not work properly. (Bug#13897)
- Repeated invocation of `my_init()` and `my_end()` caused corruption of character set data and connection failure. (Bug#6536)
- When used with the `ExtractValue()` function, an XPath expression having no leading “/” character would crash the server. (Bug#16234)
- A `SELECT` from the last partition of a subpartitioned table having a `UNIQUE KEY` could crash the MySQL Server. (Bug#16907)
- A `SELECT` on a subpartitioned table having a multiple-column `PRIMARY` or `UNIQUE KEY`, and whose partitioning function used only the first column of the key, could cause `mysqld` to crash. (Bug#16901)
- Using `REPLACE INTO` on a partitioned table having a primary key would crash the server in the event of a duplicate key error. (Bug#16782)
- `DROP TABLE` would sometimes fail on a table having subpartitions that used the default storage engine. (Bug#16775)
- **NDB Cluster**: Sharing of table names containing special characters between multiple SQL nodes was not handled correctly when binary logging was enabled (a timeout error resulted). (Bug#17415)
- **NDB Cluster**: Table definitions were not shared between multiple SQL nodes in a cluster without binary logging being enabled. (Bug#17414)
- **NDB Cluster**: Cluster log file paths were truncated to 128 characters. They may now be as long as `MAX_PATH` (the maximum path length permitted by the operating system). (Bug#17411)
- `SHOW CREATE EVENT` displayed no output. (Bug#16423)
- Statements that contained Unicode characters were not logged to the log tables correctly. (Bug#16905)
- On Windows platforms, some attempts to create partitioned tables from the command line would cause the `mysql` client to hang. (Bug#17082)
- **NDB Cluster**: `SHOW CREATE TABLE` would fail when run against a table created in a different session. (Bug#17340)
- **NDB Cluster**: Following multiple forced shutdowns and restarts of data nodes, `DROP DATABASE` could fail. (Bug#17325)
- **NDB Cluster**: An `UPDATE` with an inner join failed to match any records if both tables in the join did not have a primary key. (Bug#17257)
- **NDB Cluster**: A `DELETE` with a join in the `WHERE` clause failed to retrieve any records if both tables in the join did not have a primary key. (Bug#17249)

- The **NDB Cluster** storage engine did not allow views to be updated. (Bug#17206)
- **NDB Cluster**: Row-based replication of a cluster failed to take `--binlog_ignore_db` settings into account. (Bug#17188)
- Trying to create a partitioned table with more than 32 attributes failed. (Bug#17179)
- **NDB Cluster**: When attempting to import data into an **NDB** table using `LOAD DATA INFILE`, the server would hang in the event of a duplicate key error. (Bug#17154)
- **NDB Cluster**: In some cases, `LOAD DATA INFILE` did not load all data into **NDB** tables. (Bug#17081)
- **NDB Cluster**: Performing large numbers of data manipulation statements on cluster tables using Disk Data could lead to a server crash. ()
- **NDB Cluster**: In some cases, a cluster using Disk Data tables could not be restarted following a normal shutdown. (Bug#16872)
- **NDB Cluster**: The **REDO** log would become corrupted (and thus unreadable) in some circumstances, due to a failure in the query handler. (Bug#17295)
- **NDB Cluster**: `CREATE TABLE new_tbl LIKE old_tbl;` failed when `old_tbl` used the **NDB** storage engine. (Bug#17005)
- **NDB Cluster**: No error message was generated for setting `NoOfFragmentLogFiles` too low. (Bug#13966)
- **NDB Cluster**: No error message was generated for setting `MaxNoOfAttributes` too low. (Bug#13965)
- The **SELECT** privilege was required for triggers that performed no selects. (Bug#15196)
- The **UPDATE** privilege was required for triggers that performed no updates. (Bug#15166)
- `CAST( ... AS TIME)` operations returned different results when using versus not using prepared-statement protocol. (Bug#15805)
- Killing a long-running query containing a subquery could cause a server crash. (Bug#14851)
- **InnoDB** could display an incorrect error message for a cascading update. (Bug#9680)
- A **RETURN** statement within a trigger caused a server crash. **RETURN** now is disallowed within triggers. To exit immediately, use **LEAVE**. (Bug#16829)

C.1.24. Changes in release 5.1.6 (01 February 2006)

Functionality added or changed:

- **Packaging changes**: MySQL 5.1.6 introduces some changes to distribution packaging:
 - Distributions include both a `mysqld` optimized server and `mysqld-debug` debugging server. There is no separate debug distribution.
 - There is no longer a `mysqld-max` server. (Note: This changed in MySQL 5.1.9: The `mysqld-max` server also is included in binary distributions.)
 - Server binaries no longer are stripped, except for RPM distributions.
 - Binary distributions for Unix and Unix-like systems no longer include `safe_mysqld` as a link to `mysqld_safe`. `safe_mysqld` has been deprecated since MySQL 4.0 and now is removed.
- **Incompatible change**: This release introduces the **TRIGGER** privilege. Previously, the **SUPER** privilege was needed to create or drop triggers. Now those operations require the **TRIGGER** privilege. This is a security improvement because you no longer need to grant users the **SUPER** privilege to enable them to create triggers. However, the requirement that the account named in a trigger's **DEFINER** clause must have the **SUPER** privilege has changed to a requirement for the **TRIGGER** privilege. After upgrading, be sure to update your grant tables as described in [Section 5.5.3, “mysql_fix_privilege_tables — Upgrade MySQL System Tables”](#). This process assigns the **TRIGGER** privilege to all accounts that had the **SUPER** privilege. (After updating, you might also

consider whether any of those accounts no longer need [SUPER](#).) If you fail to update the grant tables, triggers may fail when activated. ([Bug#9142](#))

- **Incompatible change:** Before MySQL 5.1.6, the server writes general query log and slow query log entries to log files. As of MySQL 5.1.6, the server's logging capabilities for these logs are more flexible. Log entries can be written to log files (as before) or to the [general_log](#) and [slow_log](#) tables in the [mysql](#) database. If logging is enabled, either or both destinations can be selected. The [--log-output](#) option controls the destination or destinations of log output. See [Section 5.11.1, “Selecting General Query and Slow Query Log Output Destinations”](#).

If logging is enabled, the default destination now is to log to tables, which differs from earlier versions. If you had the server configured for logging to log files formerly, use [--log-output=FILE](#) to preserve this behavior after an upgrade to MySQL 5.1.6 or higher.

- **Incompatible change:** Due to a change in the naming scheme for partitioning and subpartitioning files, it is not possible for the server to read partitioned tables created in previous MySQL versions. A suggested workaround is (1) to create a non-partitioned table with the same table schema using a standard [CREATE TABLE](#) statement (that is, with no partitioning clauses) and then (2) to issue a [SELECT INTO](#) to copy the data into the non-partitioned table before the upgrade; following the upgrade, you can partition the new table using [ALTER TABLE ... PARTITION BY ...](#). Alternatively, you can dump the table using [mysqldump](#) prior to upgrading and reload it afterwards with [LOAD DATA](#). In either case, you should drop the pre-5.1.6 partitioned tables before upgrading to 5.1.6 or later. ([Bug#13437](#))

Important

If any partitioned tables that were created prior to MySQL 5.1.6 are present following an upgrade to MySQL 5.1.6 or later, it is also not possible to read from the [INFORMATION_SCHEMA.PARTITIONS](#) table, nor will you be able to drop those tables or the database or databases in which they are located. In this event, you must: (1) shut down [mysqld](#); (2) manually delete the table, partition, and (if any) subpartition files; and then (3) restart the MySQL Server. ([Bug#16695](#))

- **Incompatible change:** Words with apostrophes are now matched in a [FULLTEXT](#) search against non-apostrophe words (for example, a search for [Jerry](#) will match against the term [Jerry's](#)). Users upgrading to this version must issue [REPAIR TABLE](#) statements for tables containing [FULLTEXT](#) indexes. ([Bug#14194](#))
- MySQL 5.1.6 introduces the *Event Scheduler* which allows one to schedule statements for execution at predetermined times. Events can be transient (one-time-only) or recurrent at regular intervals, and may execute queries and statements permitted in stored routines, including compound statements.

Events can be altered after creation, and dropped when no longer needed.

Information about scheduled events can be obtained using the statements [SHOW EVENTS](#) and [SHOW CREATE EVENT](#), or by querying the [INFORMATION_SCHEMA.EVENTS](#) table. All of these are available beginning in MySQL 5.1.6.

Users must have the [EVENT](#) privilege (also added in 5.1.6) to create events.

For more information, see [Chapter 21, Event Scheduler](#).

- Replication between MySQL Clusters is now supported. It is now also possible to replicate between a MySQL Cluster and a non-cluster database. See [Section 16.10, “MySQL Cluster Replication”](#).
- Special characters in database and table identifiers now are encoded when creating the corresponding directory names and filenames. This relaxes the restrictions on the characters that can appear in identifiers. See [Section 8.2.3, “Mapping of Identifiers to Filenames”](#).
- Queries against partitioned tables can now take advantage of partition pruning. In some cases, this can result in query execution that is an order of magnitude faster than the same query against a non-partitioned version of the same table.
- The [mysqldump](#) utility now supports an option for dumping tablespaces. Use [-Y](#) or [--all-tablespaces](#) to enable this functionality. ([Bug#16753](#))
- Partition support is not an “engine”, but it was included in the output of [SHOW ENGINES](#). Now it is not. ([Bug#14355](#)) The [have_partition_engine](#) variable was renamed to [have_partitioning](#). ([Bug#16718](#))
- [ANALYZE TABLE](#) is now supported for partitioned tables. ([Bug#13441](#))
- Added the [event_scheduler](#) system variable.
- Added the [ndb_extra_logging](#) system variable.

- Added the `FILES` table to `INFORMATION_SCHEMA`.
- Added the `EVENTS` table to `INFORMATION_SCHEMA`.
- Added the `PARTITIONS` table to `INFORMATION_SCHEMA`.
- The `ARCHIVE` storage engine now supports the `AUTO_INCREMENT` column attribute and the `AUTO_INCREMENT` table option. [Section 13.10, “The ARCHIVE Storage Engine”](#).
- Added support for the `CREATE INDEX` and `DROP INDEX` statements to the `NDB Cluster` storage engine.
- Server plugins can register their own status variables to be displayed by the `SHOW STATUS` statement.

Bugs fixed:

- An indexing error sometimes caused values to be assigned to the wrong `RANGE` partition. ([Bug#16684](#))
- `NDB Cluster`: `ndb_delete_all` would run out of memory on tables containing `BLOB` columns. ([Bug#16693](#))
- Using the `TRUNCATE()` function with a negative number for the second argument on a `BIGINT` column returned incorrect results. ([Bug#8461](#))
- When the fulltext search parser plugin returned more words than half of the length (in bytes) of the query string, the server would crash. ([Bug#16722](#))
- Improper memory handling for stored routine variables could cause memory overruns and binary log corruption. ([Bug#15588](#))
- A `FULLTEXT` query in a prepared statement could result in unexpected behavior. ([Bug#14496](#))
- `STR_TO_DATE(1, NULL)` caused a server crash. ([CVE-2006-3081](#), [Bug#15828](#))
- An `INSERT` statement in a stored procedure corrupted the binary log. ([Bug#16621](#))
- The `mysql_real_connect()` C API function incorrectly reset the `MYSQL_OPT_RECONNECT` option to its default value. ([Bug#15719](#))
- Specifying a value for `--tmpdir` without a trailing slash had unpredictable results. ([Bug#15904](#))
- Attempting to insert data into a partitioned table that used the `BLACKHOLE` storage engine caused `mysqld` to crash. ([Bug#14524](#))
- Using `RANGE` partitioning with a `CASE` statement as the partitioning function would cause records to be placed in the wrong partition. ([Bug#15393](#))
- `ALTER TABLE ... ADD PARTITIONS` on a table with one partition crashed the server. ([Bug#15820](#))
- `NDB Cluster` returned incorrect `Can't find file` error for OS error 24, changed to `Too many open files`. ([Bug#15020](#))
- Multi-byte path names for `LOAD DATA` and `SELECT ... INTO OUTFILE` caused errors. Added the `character_set_filesystem` system variable, which controls the interpretation of string literals that refer to filenames. ([Bug#12448](#))
- Certain subqueries where the inner query is the result of an aggregate function would return different results on MySQL 5.0 than on MySQL 4.1. ([Bug#15347](#))
- The error message for specifying values for which no partition exists returned wrong values on certain platforms. ([Bug#15910](#))
- Certain Japanese table names were not properly saved during a `CREATE TABLE` statement. ([Bug#3906](#))
- `NDB Cluster` leaked disk space when performing `INSERTS/DELETES` in a loop. ([Bug#16771](#))
- `NDB Cluster` returned wrong error when tablespace on disk was full. ([Bug#16738](#))
- The `DATA DIRECTORY` and `INDEX DIRECTORY` clauses of a `CREATE TABLE` statement involving partitions did not work. ([Bug#14354](#))
- Subselect could return wrong results when records cache and grouping was involved. ([Bug#15347](#))

- In some cases the query optimizer did not properly perform multiple joins where inner joins followed left joins, resulting in corrupted result sets. (Bug#15633)
- The absence of a table in the left part of a left or right join was not checked prior to name resolution, which resulted in a server crash. (Bug#15538)
- **NDB Cluster**: Trying to import too many dumped tables requiring resources beyond those allocated in the cluster configuration would cause the server to crash instead of reporting an insufficient resources error. (Bug#16455)
- **NDB Cluster** (Disk Data): Tablespaces created using parameters with relatively low values (< 10 MB) produced filesizes much smaller than expected. (Bug#16742)
- **NDB Cluster**: **CREATE TABLESPACE** statements were incorrectly parsed on 64-bit platforms. (**INITIAL SIZE size** worked, but **INITIAL SIZE = size** failed.) (Bug#13556)
- Trying to add more than one partition in a single **ALTER TABLE ... ADD PARTITION** statement caused the server to crash. (Bug#16534)
- Creating a partitioned table using a storage engine other than the session default storage engine caused the server to crash. (Bug#15966)
- An **ALTER TABLE ... PARTITION BY ...** statement did not have any effect. (Bug#15523)
- **NDBCluster** (Disk Data): The error message generated by a failed **ADD UNDOFILE** did not provide any reasons for the failure. (Bug#16267)
- **NDBCluster** (Disk Data): **DROP LOGFILE GROUP** corrupted the cluster file system and caused **ndbd** to fail when running more than one node on the same system. (Bug#16193)
- **NDBCluster**: A bitfield whose offset and length totaled 32 would crash the cluster. (Bug#16125)
- **NDBCluster**: Upon the completion of a scan where a key request remained outstanding on the primary replica and a starting node died, the scan did not terminate. This caused incompleted error handling of the failed node. (Bug#15908)
- **NDBCluster**: The **ndb_autodiscover** test failed sporadically due to a node not being permitted to connect to the cluster. (Bug#15619)
- Using **mysqldump** to obtain a dump of a partitioned table employing the **NDB** storage engine produced a non-functional table creation statement. (Bug#13155)
- **SHOW CREATE TABLE** did not display the **PARTITIONS** clause for tables partitioned by **HASH** or **KEY**. (Bug#14327)
- Inserting a negative value into an integer column used as the partitioning key for a table partitioned by **HASH** could cause the server to crash. (Bug#15968)
- With a table partitioned by **LIST**, inserting a value which was smaller than any value shown in the partitioning value-lists could cause the server to crash. (Bug#14365)
- **ALTER TABLE ... DROP PARTITION** would truncate all **DATE** column values in the table's remaining partitions to **NULL**. (Bug#13644)
- **ALTER TABLE ... ADD PARTITION** could crash the server or cause an **OUT OF MEMORY** error in some circumstances. (Bug#13447)
- The server would allow foreign keys to be declared in the definition of a partitioned table despite the fact that partitioned tables do not support foreign keys (see [Section 17.5, “Restrictions and Limitations on Partitioning”](#)). (Bug#13446)
- A **SELECT** from a key-partitioned table with a multi-column key could cause the server to crash. (Bug#13445)
- Issuing a **TRUNCATE** statement twice in succession on the same partitioned table would cause the server to crash. (Bug#13442)
- Using a **REPLACE** statement on a partitioned table caused the server to crash. (Bug#13440)
- Using an identifier rather than a literal integer value in the **LESS THAN** clause of a range-partitioned table could cause the server to crash and corruption of tables. (Bug#13439)

- Using `ENGINE=...` within a `PARTITION` clause could cause the server to crash. ([Bug#13438](#))
- `CREATE TABLE ... LIKE` did not work if the table whose schema was to be copied was a partitioned table. ([Bug#13435](#))
- `SHOW CREATE TABLE` did not display the `PARTITIONS` clause for tables partitioned by `HASH` or `KEY`. ([Bug#14327](#))
- Certain permission management statements could create a `NULL` hostname for a user, resulting in a server crash. ([Bug#15598](#))
- Temporary table aliasing did not work inside stored functions. ([Bug#12198](#))
- Parallel builds occasionally failed on Solaris. ([Bug#16282](#))

C.1.25. Changes in release 5.1.5 (10 January 2006)

Functionality added or changed:

- Added the `INFORMATION_SCHEMA ENGINES` table.
- Added the `INFORMATION_SCHEMA PLUGINS` table and the `SHOW PLUGIN` statement.
- Added the `binlog_format` system variable that controls whether to use row-based or statement-based binary logging. Added the `--binlog-format` and `--binlog-row-event-max-size` server options for binary logging control. See [Section 15.1.2](#), “Replication Formats”.
- Plugins now can have status variables that are displayed in the output from `SHOW STATUS`. See [Section 28.2.5](#), “Writing Plugins”.
- Added the XML functions `ExtractValue()` and `UpdateXML()`. `ExtractValue()` returns the content of a fragment of XML matching a given XPath expression. `UpdateXML()` replaces the element selected from a fragment of XML by an XPath expression supplied by the user with a second XML fragment (also user-supplied), and returns the modified XML. See [Section 11.10](#), “XML Functions”.
- Added the `--base64-output` option to `mysqlbinlog` to print all binary log entries using base64 encoding. This is for debug-only. Logs produced using this option should not be applied on production systems.
- Added the `--port-open-timeout` option to `mysqld` to control how many seconds the server should wait for the TCP/IP port to become free if it cannot be opened. ([Bug#15591](#))
- Two new Hungarian collations are included: `utf8_hungarian_ci` and `ucs2_hungarian_ci`. These support the correct sort order for Hungarian vowels. However, they do not support the correct order for sorting Hungarian consonant contractions; this issue will be fixed in a future release.

Bugs fixed:

- **InnoDB**: An `UPDATE` statement with no index column in the `WHERE` condition locked all the rows in the table. ([Bug#3300](#))
- `INSERT DELAYED` caused `mysqld` to crash. ([Bug#16095](#))
- The output of `mysqldump --triggers` did not contain the `DEFINER` clause in dumped trigger definitions. ([Bug#15110](#))
- The output of `SHOW TRIGGERS` contained extraneous whitespace. ([Bug#15103](#))
- An `INSERT ... SELECT` statement between tables in a `MERGE` set can return errors when statement involves insert into child table from merge table or vice-versa. ([Bug#5390](#))
- A `COMMIT` statement followed by a `ALTER TABLE` statement on a BDB table caused server crash. ([Bug#14212](#))
- **InnoDB**: Comparison of indexed `VARCHAR CHARACTER SET ucs2 COLLATE ucs2_bin` columns using `LIKE` could fail. ([Bug#14583](#))
- Creating a trigger caused a server crash if the table or trigger database was not known because no default database had been selected. ([Bug#14863](#))

- Issuing a `DROP USER` command could cause some users to encounter a `hostname is not allowed to connect to this MySQL server` error. (Bug#15775)
- The `--plugin_dir` option was not working. Also fix error with specifying parser name for fulltext. (Bug#16068)
- Attempting to insert into a table partitioned by `LIST` a value less than any specified in one of the table's partition definitions resulted in a server crash. In such cases, `mysqld` now returns `ERROR 1500 (HY000): TABLE HAS NO PARTITION FOR VALUE v`, where `v` is the out-of-range value. (Bug#15819)

C.1.26. Changes in release 5.1.4 (21 December 2005)

Functionality added or changed:

- Added the `mysqlslap` program, which is designed to emulate client load for a MySQL server and report the timing of each stage. It works as if multiple clients are accessing the server.
- Added the `--server-id` option to `mysqlbinlog` to enable only those events created by the server having the given server ID to be extracted. (Bug#15485)
- It is now possible to build the server such that `MyISAM` tables can support up to 128 keys rather than the standard 64. This can be done by configuring the build using the option `--with-max-indexes=N`, where `N#128` is the maximum number of indexes to permit per table. (Bug#10932)
- The bundled `BDB` library was upgraded to version 4.4.16.
- Added the `cp1250_polish_ci` collation for the `cp1250` character set.
- Added the `myisam_use_mmap` system variable.
- Added the `--bdb-data-direct` and `--bdb-log-direct` server options.

Bugs fixed:

- `SHOW ENGINES` output showed the `FEDERATED` engine as `DISABLED` even for builds with `FEDERATED` support. (Bug#15559)
- Server could not be built on default Debian systems with `BDB` enabled. (Bug#15734)
- `BDB`: A `DELETE`, `INSERT`, or `UPDATE` of a `BDB` table could cause the server to crash where the query contained a subquery using an index read. (Bug#15536)
- A left join on a column that having a `NULL` value could cause the server to crash. (Bug#15268)
- It was not possible to reorganize a partition reusing a discarded partition name.

Now, for example, you can create a table such as this one:

```
CREATE TABLE t1 (a INT)
PARTITION BY RANGE (a) (
  PARTITION p0 VALUES LESS THAN (10),
  PARTITION p1 VALUES LESS THAN (20),
  PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

and then repartition it as shown here:

```
ALTER TABLE t1 REORGANIZE PARTITION p2 INTO (
  PARTITION p2 VALUES LESS THAN (30)
);
```

Previously, attempting to do so would produce the error `ALL PARTITIONS MUST HAVE UNIQUE NAMES IN THE TABLE`. (Bug#15521)

- `NDB Cluster`: The `--ndb` option for `percona` did not function. (Bug#15486)

- The **BLACKHOLE** storage engine did not handle transactions properly: Rolled-back transactions were written to the binary log. Now they are not. ([Bug#15406](#))
- **NDB Cluster**: Using **ORDER BY primary_key_column** when selecting from a table having the primary key on a **VARCHAR** column caused a forced shutdown of the cluster. ([Bug#14828](#), [Bug#15240](#), [Bug#15682](#), [Bug#15517](#))
- **ANALYZE TABLE** did not properly update table statistics for a **MyISAM** table with a **FULLTEXT** index containing stopwords, so a subsequent **ANALYZE TABLE** would not recognize the table as having already been analyzed. ([Bug#14902](#))
- The maximum value of **MAX_ROWS** was handled incorrectly on 64-bit systems. ([Bug#14155](#))
- Multiple-table update operations were counting updates and not updated rows. As a result, if a row had several updates it was counted several times for the “rows matched” value but updated only once. ([Bug#15028](#))
- **SELECT** queries that began with an opening parenthesis were not being placed in the query cache. ([Bug#14652](#))
- Space truncation was being ignored when inserting into **BINARY** or **VARBINARY** columns. Now space truncation results in a warning, or an error in strict mode. ([Bug#14299](#))
- Selecting from a view processed with the temptable algorithm caused a server crash if the query cache was enabled. ([Bug#15119](#))
- Creating a view that referenced a stored function that selected from a view caused a crash upon selection from the view. ([Bug#15096](#))
- Creating a view within a stored procedure could result in an out of memory error or a server crash. ([Bug#14885](#))
- **SHOW CREATE DATABASE** was sometimes refused when the client had privileges for the database. ([Bug#9785](#))
- **mysql** ignored the **MYSQL_TCP_PORT** environment variable. ([Bug#5792](#))
- **ROW_COUNT()** returned an incorrect result after **EXECUTE** of a prepared statement. ([Bug#14956](#))
- Invalid casts to **DATE** values now result in a message of **Incorrect datetime value**, rather than **Truncated incorrect datetime value**. ([Bug#8294](#))
- Attempts to assign **NULL** to a **NOT NULL** column in strict mode now result in a message of **Column 'col_name' cannot be null, rather than Column set to default value; NULL supplied to NOT NULL column 'col_name' at row n**. ([Bug#11491](#))
- For binary string data types, **mysqldump --hex-blob** produced an illegal output value of **0x** rather than **'**. ([Bug#13318](#))
- Some comparisons for the **IN()** operator were inconsistent with equivalent comparisons for the **=** operator. ([Bug#12612](#))

C.1.27. Changes in release 5.1.3 (29 November 2005)

Functionality added or changed:

This is the first public alpha release of the current MySQL 5.1 development branch, providing an insight to upcoming features. Although some of these are still under heavy development, this release includes the following new features and changes (in comparison to the current MySQL 5.0 production release):

- **Partitioning**: allows distributing portions of individual tables across a filesystem, according to rules which can be set when the table is created. In effect, different portions of a table are stored as separate tables in different locations, but from the user point of view, the partitioned table is still a single table. See [Chapter 17, Partitioning](#), for further information on this functionality. (Author: Mikael Ronström)
- **Plugin API**: MySQL 5.1 adds support for a very flexible plugin API that enables loading and unloading of various components at runtime, without restarting the server. Although the work on this is not finished yet, *plugin full-text parsers* are a first step in this direction. This allows users to implement their own input filter on the indexed text, enabling full-text search capability on arbitrary data such as PDF files or other document formats. A pre-parser full-text plugin performs the actual parsing and extraction of the text and hands it over to the built-in MySQL full-text search. (Author: Sergey Vojtovich)

The plugin API requires the **mysql.plugin** table. When upgrading from an older version of MySQL, you should run the **mysql_fix_privilege_tables** command to create this table. See [Section 5.5.3, “mysql_fix_privilege_tables —](#)

Upgrade MySQL System Tables”.

Incompatible change: Plugins are installed in the directory named by the `plugin_dir` system variable. This variable also controls the location from which the server loads user-defined functions (UDFs), which is a change from earlier versions of MySQL. That is, all UDF library files now must be installed in the plugin directory. When upgrading from an older version of MySQL, you must migrate your UDF files to the plugin directory.

- **The Instance Manager (IM)** now has some additional functionality:
 - `SHOW instance_name LOG FILES` provides a listing of all log files used by the instance. (Author: Petr Chardin)
 - `SHOW instance_name LOG {ERROR | SLOW | GENERAL} size` retrieves a part of the specified log file. (Author: Petr Chardin)
 - `SET instance_name.option_name=option_value` sets an option to the specified value and writes it to the config file. See [Section 5.4, “mysqlmanager — The MySQL Instance Manager”](#), for more details on these new commands. (Author: Petr Chardin)
- The performance of boolean full-text searches (using the “+” Operator) has been improved. See [Section 11.8, “Full-Text Search Functions”](#), for more details about full-text searching. (Author: Sergey Vojtovich)
- `VARCHAR` fields used in MySQL Cluster tables are now variable-sized; that is, they now only allocate as much space as required to store the data. Previously, a `VARCHAR(n)` column allocated $n+2$ bytes (aligned to 4 bytes), regardless if the actual inserted value required that much space. (In other words, a `VARCHAR` column always required the same, fixed, amount of storage as a `CHAR` column of the same size.)
- Renamed the `table_cache` system variable to `table_open_cache`. Any scripts that refer to `table_cache` should be updated to use the new name.
- Added the `table_definition_cache` system variable. If you use a large number of tables, you can create a large table definition cache to speed up opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the normal table cache.
- Added the `SHOW AUTHORS` statement.
- Added the `SHOW FUNCTION CODE` and `SHOW PROCEDURE CODE` statements (available only for servers that have been built with debugging support). See [Section 12.5.4.23, “SHOW PROCEDURE CODE and SHOW FUNCTION CODE Syntax”](#).
- `RAND( )` no longer allows non-constant initializers. (Prior to MySQL 5.0.13, the effect of non-constant initializers is undefined.) ([Bug#6172](#))

Bugs fixed:

- Set functions could not be aggregated in outer subqueries. ([Bug#12762](#))

C.1.28. Changes in release 5.1.2 (Not released)

Functionality added or changed:

- Added `MAXLOCKS`, `MINLOCKS`, `MAXWRITE`, and `MINWRITE` as allowable values of the `--bdb-lock-detect` option. ([Bug#14876](#))
- Added the `bdb_cache_parts` and `bdb_region_size` system variables, and allowed `bdb_cache_size` to be larger than 4GB on systems that support it. ([Bug#14895](#))
- Added `Transactions`, `XA`, and `Savepoints` columns to `SHOW ENGINES` output.
- Added `--replace` to `mysqldump`. This option uses `REPLACE INTO`, rather than `INSERT INTO`, when writing the dumpfile.

Bugs fixed:

- Foreign keys were not properly enforced in [TEMPORARY](#) tables. Foreign keys now are disallowed in [TEMPORARY](#) tables. ([Bug#12084](#))

C.1.29. Changes in release 5.1.1 (Not released)

Bugs fixed:

- Performing a [CREATE TABLE](#) statement with a [PARTITION BY](#) clause in a prepared statement could crash a server running in debug mode. ([Bug#12097](#))
- [NDB](#): Specifying the wrong nodegroup in a [CREATE TABLE](#) using partitioning would lead to the table name being locked after the [CREATE TABLE](#) statement failed (that is, the table name could not be re-used). ([Bug#12114](#))
- Using [ORDER BY](#) in a query with a partitioned table on a 64-bit operating system could crash the server. ([Bug#12116](#))
- When two threads competed for the same table, a deadlock could occur if one thread also had a lock on another table through [LOCK TABLES](#) and the thread was attempting to remove the table in some manner while the other thread tried to place locks on both tables. ([Bug#10600](#))

C.2. MySQL Connector/ODBC (MyODBC) Change History

C.2.1. Changes in Connector/ODBC 5.0.12 (Not yet released)

Bugs fixed:

- Tables with [TEXT](#) columns would be incorrectly identified, returning an [Unknown SQL type - 65535](#) error. ([Bug#20127](#))
- Inserting [NULL](#) values into a [DATETIME](#) column from Access reports an error. ([Bug#27896](#))

C.2.2. Changes in Connector/ODBC 5.0.11 (31 January 2007)

Functionality added or changed:

- Driver now builds and is partially tested under Linux with the iODBC driver manager.
- Added support for ODBC v2 statement options using attributes.

Bugs fixed:

- Internal function, [my_setpos_delete_ignore\(\)](#) could cause a crash. ([Bug#22796](#))
- Updates of [MEMO](#) or [TEXT](#) columns from within Microsoft Access would fail. ([Bug#25263](#))
- Transaction support has been added and tested. ([Bug#25045](#))
- Connection string parsing for DSN-less connections could fail to identify some parameters. ([Bug#25316](#))
- Fixed occasional mis-handling of the [SQL_NUMERIC_C](#) type.
- Fixed the binding of certain integer types.

C.2.3. Changes in Connector/ODBC 5.0.10 (14 December 2006)

Connector/ODBC 5.0.10 is the sixth BETA release.

Functionality added or changed:

- Added wide-string type info for `SQLGetTypeInfo()`.
- Added loose handling of retrieving some diagnostic data. ([Bug#15782](#))
- Added initial support for removing braces when calling stored procedures and retrieving result sets from procedure calls. ([Bug#24485](#))
- Added initial unicode support in data and metadata. ([Bug#24837](#))
- Significant performance improvement when retrieving large text fields in pieces using `SQLGetData()` with a buffer smaller than the whole data. Mainly used in Access when fetching very large text fields. ([Bug#24876](#))

Bugs fixed:

- String query parameters are now escaped correctly. ([Bug#19078](#))
- Editing DSN no longer crashes ODBC data source administrator. ([Bug#24675](#))

C.2.4. Changes in Connector/ODBC 5.0.9 (22 November 2006)

Connector/ODBC 5.0.9 is the fifth BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Added recognition of `SQL_C_SHORT` and `SQL_C_TINYINT` as C types.
- Added support for column binding as `SQL_NUMERIC_STRUCT`.

Bugs fixed:

- Fixed wildcard handling of and listing of catalogs and tables in `SQLTables`.
- Catch use of `SQL_ATTR_PARAMSET_SIZE` and report error until we fully support.
- Corrected retrieval multiple field types bit and blob/text.
- Fixed buffer length return for `SQLDriverConnect`.
- Added limit of display size when requested via `SQLColAttribute/SQL_DESC_DISPLAY_SIZE`.
- Fixed statistics to fail if it couldn't be completed.
- Fixed `SQLGetData` to clear the NULL indicator correctly during multiple calls.
- ODBC v2 behaviour in driver now supports ODBC v3 date/time types (since `DriverManager` maps them).

C.2.5. Changes in Connector/ODBC 5.0.8 (17 November 2006)

Connector/ODBC 5.0.8 is the fourth BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Made distinction between [CHAR/BINARY](#) (and VAR versions).
- Wildcards now support escaped chars and underscore matching (needed to link tables with underscores in access).
- Also made [SQL_DESC_NAME](#) only fill in the name if there was a data pointer given, otherwise just the length.
- Fixed display size to be length if max length isn't available.

Bugs fixed:

- Length now used when handling bind parameter (needed in particular for [SQL_WCHAR](#)) - this enables updating char data in MS Access.
- Fixed string length to chars, not bytes, returned by [SQLGetDiagRec](#).
- Fixed using wrong pointer for [SQL_MAX_DRIVER_CONNECTIONS](#) in [SQLGetInfo](#).
- Fixed binding using [SQL_C_LONG](#).
- Allow [SQLDescribeCol](#) to be called to retrieve the length of the column name, but not the name itself.
- Fix size return from [SQLDescribeCol](#).
- Fixed handling of numeric pointers in [SQLColAttribute](#).
- Fixed type returned for [MYSQL_TYPE_LONG](#) to [SQL_INTEGER](#) instead of [SQL_TINYINT](#).
- Fixed [MDiagnostic](#) to use correct v2/v3 error codes.
- Set default return to [SQL_SUCCESS](#) if nothing is done for [SQLSpecialColumns](#).
- Updated retrieval of descriptor fields to use the right pointer types.

C.2.6. Changes in Connector/ODBC 5.0.7 (08 November 2006)

Connector/ODBC 5.0.7 is the third BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Improved trace/log.
- Added support for [SQLStatistics](#) to [MYODBCShell](#).

Bugs fixed:

- Fixed [SQLDescribeCol](#) returning column name length in bytes rather than chars.
- [SQLBindParameter](#) now handles [SQL_C_DEFAULT](#).
- Corrected incorrect column index within [SQLStatistics](#). Many more tables can now be linked into MS Access.

C.2.7. Changes in Connector/ODBC 5.0.6 (03 November 2006)

Connector/ODBC 5.0.6 is the second BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release:

- You no longer have to have Connector/ODBC 3.51 installed before installing this version.
- Installation is provided in the form of a standard Microsoft System Installer (MSI).
- Connector/ODBC supports both [User](#) and [System](#) DSNs.

C.2.8. Changes in Connector/ODBC 5.0.5 (17 October 2006)

Connector/ODBC 5.0.5 is the first BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release:

- You no longer have to have Connector/ODBC 3.51 installed before installing this version.

C.2.9. Changes in Connector/ODBC 5.0.3 (Connector/ODBC 5.0 Alpha 3) (20 June 2006)

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release:

- The following ODBC API functions have been added in this release:
 - [SQLBindParameter](#)
 - [SQLBindCol](#)

C.2.10. Changes in Connector/ODBC 5.0.2 (Never released)

Connector/ODBC 5.0.2 was an internal implementation and testing release.

C.2.11. Changes in Connector/ODBC 5.0.1 (Connector/ODBC 5.0 Alpha 2) (05 June 2006)

Features, limitations and notes on this release:

- Connector/ODBC 5.0 is Unicode aware.
- Connector/ODBC is currently limited to basic applications. ADO applications and Microsoft Office are not supported.
- Connector/ODBC must be used with a Driver Manager.
- The following ODBC API functions are implemented:
 - [SQLAllocHandle](#)
 - [SQLCloseCursor](#)
 - [SQLColAttribute](#)
 - [SQLColumns](#)
 - [SQLConnect](#)

- [SQLCopyDesc](#)
- [SQLDisconnect](#)
- [SQLExecDirect](#)
- [SQLExecute](#)
- [SQLFetch](#)
- [SQLFreeHandle](#)
- [SQLFreeStmt](#)
- [SQLGetConnectAttr](#)
- [SQLGetData](#)
- [SQLGetDescField](#)
- [SQLGetDescRec](#)
- [SQLGetDiagField](#)
- [SQLGetDiagRec](#)
- [SQLGetEnvAttr](#)
- [SQLGetFunctions](#)
- [SQLGetStmtAttr](#)
- [SQLGetTypeInfo](#)
- [SQLNumResultCols](#)
- [SQLPrepare](#)
- [SQLRowcount](#)
- [SQLTables](#)

The following ODBC API function are implemented, but not yet support all the available attributes/options:

- [SQLSetConnectAttr](#)
- [SQLSetDescField](#)
- [SQLSetDescRec](#)
- [SQLSetEnvAttr](#)
- [SQLSetStmtAttr](#)

C.2.12. Changes in Connector/ODBC 3.51.19 (10 August 2007)

Connector/ODBC 3.5.19 fixes a specific issue with the 3.5.18 release. For a list of changes in the 3.5.18 release, see [Section C.2.13, “Changes in Connector/ODBC 3.51.18 \(08 August 2007\)”](#).

Functionality added or changed:

- Because of [Bug#10491](#) in the server, character string results were sometimes incorrectly identified as [SQL_VARBINARY](#). Until this server bug is corrected, the driver will identify all variable-length strings as [SQL_VARCHAR](#).

C.2.13. Changes in Connector/ODBC 3.51.18 (08 August 2007)

Platform specific notes:

- A binary package without an installer is available for Microsoft Windows x64 Edition. There are no installer packages for Microsoft Windows x64 Edition.
- Binary packages as disk images with installers are now available for Mac OS X.
- Binary packages for Sun Solaris are now available as [PKG](#) packages.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.

Functionality added or changed:

- Connector/ODBC now supports batched statements. In order to enable cached statement support you must switch enable the batched statement option ([FLAG_MULTI_STATEMENTS](#), 67108864, or [ALLOW_MULTIPLE_STATEMENTS](#) within a GUI configuration). Be aware that batched statements create an increased chance of SQL injection attacks and you must ensure that your application protects against this scenario. ([Bug#7445](#))
- The [SQL_ATTR_ROW_BIND_OFFSET_PTR](#) is now supported for row bind offsets. ([Bug#6741](#))
- When connecting to a specific database when using a DSN, the system tables from the [mysql](#) database are no longer also available. Previously, tables from the [mysql](#) database (catalog) were listed as [SYSTEM TABLES](#) by [SQLTables\(\)](#) even when a different catalog was being queried. ([Bug#28662](#))
- The [TRACE](#) and [TRACEFILE](#) DSN options have been removed. Use the ODBC driver manager trace options instead.
- Installed for Mac OS X has been re-instated. The installer registers the driver at a system (not user) level and makes it possible to create both user and system DSNs using the Connector/ODBC driver. The installer also fixes the situation where the necessary drivers would bge installed local to the user, not globally. ([Bug#10444](#), [Bug#15326](#))

Bugs fixed:

- The [SQLColumns\(\)](#) function could return incorrect information about [AUTO_INCREMENT](#) columns, indicating that the field was not nullable. ([Bug#14407](#))
- The [SQLColumns\(\)](#) function could return incorrect information about [TIMESTAMP](#) columns, indicating that the field was not nullable. ([Bug#14414](#))
- Obtaining the length of a column when using a character set for the connection of [utf8](#) would result in the length being returned incorrectly. ([Bug#19345](#))
- The [SQL_COLUMN_TYPE](#), [SQL_COLUMN_DISPLAY](#) and [SQL_COLUMN_PRECISION](#) values would be returned incorrectly by [SQLColumns\(\)](#), [SQLDescribeCol\(\)](#) and [SQLColAttribute\(\)](#) when accessing character columns, especially those generated through [concat\(\)](#). The lengths returned should now conform to the ODBC specification. The [FLAG_FIELD_LENGTH](#) option no longer has any affect on the results returned. ([Bug#27862](#))
- The [SQL_DATA_TYPE](#) column in [SQLColumns\(\)](#) results did not report the correct value for date and time types.
- The [SQL_DATETIME_SUB](#) column in [SQLColumns\(\)](#) was not correctly set for date and time types.
- The value for [SQL_DESC_FIXED_PREC_SCALE](#) was not returned correctly for values in MySQL 5.0 and later.
- The wrong value from [SQL_DESC_LITERAL_SUFFIX](#) was returned for binary fields.
- The wrong value for [SQL_DESC_TYPE](#) was returned for date and time types.
- The wrong value for [DECIMAL_DIGITS](#) in [SQLColumns\(\)](#) was reported for [FLOAT](#) and [DOUBLE](#) fields, as well as the wrong

value for the scale parameter to `SQLDescribeCol()`, and the `SQL_DESC_SCALE` attribute from `SQLColAttribute()`.

- `BIT(n)` columns are now treated as `SQL_BIT` data where `n = 1` and binary data where `n > 1`.
- When using a table with multiple `TIMESTAMP` columns, the final `TIMESTAMP` column within the table definition would not be up-dateable. Note that there is still a limitation in MySQL server regarding multiple `TIMESTAMP` columns ([Bug#30081](#)). ([Bug#9927](#))
- Fixed an issue where the `myodbc3i` would update the the user ODBC configuration file (`~/Library/ODBC/odbcinst.ini`) instead of the system `/Library/ODBC/odbcinst.ini`. This was caused because `myodbc3i` was not honouring the `s` and `u` modifiers for the `-d` command line option. ([Bug#29964](#))
- `SQLConnect()` and `SQLDriverConnect()` were rewritten to eliminate duplicate code and ensure all options were supported using both connection methods. `SQLDriverConnect()` now only requires the setup library to be present when the call requires it.
- Getting table metadata (through the `SQLColumns()`) would fail, returning a bad table definition to calling applications. ([Bug#29888](#))
- `DATETIME` column types would return `FALSE` in place of `SQL_SUCCESS` when requesting the column type information. ([Bug#28657](#))

C.2.14. Changes in Connector/ODBC 3.51.17 (14 July 2007)

Platform specific notes:

- There are no binary packages for Microsoft Windows x64 Edition.
- The Mac OS X binary packages are only provided as tarballs, there is no installer.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.
- The binary packages for Sun Solaris are only provided as tarballs, not the PKG format.

Functionality added or changed:

- It is now possible to specify a different character set as part of the DSN or connection string. This must be used instead of the `SET NAMES` statement. You can also configure the character set value from the GUI configuration. ([Bug#6667](#), [Bug#9498](#))
- The setup library has been split into its own RPM package, to allow installing the driver itself with no GUI dependencies.
- Dis-allow NULL ptr for null indicator when calling `SQLGetData()` if value is null. Now returns `SQL_ERROR` w/state 22002.
- Fixed calling convention ptr and wrong free in `myodbc3i`, and fixed the null terminating (was only one, not two) when writing DSN to string.

Bugs fixed:

- Fixed a problem where the GUI would crash when configuring or removing a System or User DSN. ([Bug#27315](#))
- When inserting a large `BLOB` field, Connector/ODBC would crash due to a memory allocation error. ([Bug#10562](#))
- `myodbc3i` did not correctly format driver info, which could cause the installation to fail. ([Bug#29709](#))
- Added a new DSN option (`FLAG_ZERO_DATE_TO_MIN`) to retrieve `XXXX-00-00` dates as the minimum allowed ODBC date (`XXXX-01-01`). Added another option (`FLAG_MIN_DATE_TO_ZERO`) to mirror this but for bound parameters. `FLAG_MIN_DATE_TO_ZERO` only changes `0000-01-01` to `0000-00-00`. ([Bug#13766](#))

- Correctly return error if `SQLBindCol` is called with an invalid column.
- Fixed possible crash if `SQLBindCol()` was not called before `SQLSetPos()`.
- Fixed error handling of out-of-memory and bad connections in catalog functions. This might raise errors in code paths that had ignored them in the past. (Bug#26934)
- Calling `SQLGetDiagField` with `RecNumber 0`, `DiagIdentifier NOT 0` returned `SQL_ERROR`, preventing access to diagnostic header fields. (Bug#16224)
- Connector/ODBC crashed with Crystal Reports due to a problem with `SQLProcedures()`. (Bug#28316)
- For a stored procedure that returns multiple result sets, Connector/ODBC returned only the first result set. (Bug#16817)
- The driver was using `mysql_odbc_escape_string()`, which does not handle the `NO_BACKSLASH_ESCAPES` SQL mode. Now it uses `mysql_real_escape_string()`, which does. (Bug#9498)
- `SQLColumns()` did not handle many of its parameters correctly, which could lead to incorrect results. The table name argument was not handled as a pattern value, and most arguments were not escaped correctly when they contained non-alphanumeric characters. (Bug#8860)
- If there was more than one unique key on a table, the correct fields were not used in handling `SQLSetPos()`. (Bug#10563)

C.2.15. Changes in Connector/ODBC 3.51.16 (14 June 2007)

Functionality added or changed:

- Connector/ODBC now supports using SSL for communication. This is not yet exposed in the setup GUI, but must be enabled through configuration files or the DSN. (Bug#12918)

Bugs fixed:

- When using stored procedures, making a `SELECT` or second stored procedure call after an initial stored procedure call, the second statement will fail. (Bug#27544)
- Using cursors on results sets with multi-column keys could select the wrong value. (Bug#28255)
- Using `BETWEEN` with date values, the wrong results could be returned. (Bug#15773)
- Specifying strings as parameters using the `adBSTr` or `adVarWChar` types, (`SQL_WVARCHAR` and `SQL_WLONGVARCHAR`) would be incorrectly quoted. (Bug#16235)
- Return values from `SQLTables()` may be truncated. (Bugs #22797)
- Data in `TEXT` columns would fail to be read correctly. (Bug#16917)
- `SQLForeignKeys` does not escape `_` and `%` in the table name arguments. (Bug#27723)
- When using the `Don't Cache Results` (option value 1048576) with Microsoft Access, the connection will fail using DAO/VisualBasic. (Bug#4657)
- Calls to `SQLNativeSql()` could cause stack corruption due to an incorrect pointer cast. (Bug#28758)
- `SQLTables()` did not distinguish tables from views. (Bug#23031)
- `SQL_WVARCHAR` and `SQL_WLONGVARCHAR` parameters were not properly quoted and escaped. (Bug#16235)

C.2.16. Changes in Connector/ODBC 3.51.15 (7 May 2007)

Bugs fixed:

- Adding a new DSN with the `myodbc3i` utility under AIX would fail. (Bug#27220)
- Connector/ODBC would incorrectly claim to support `SQLProcedureColumns` (by returning true when queried about `SQLPROCEDURECOLUMNS` with `SQLGetFunctions`), but this functionality is not supported. (Bug#27591)
-
- Using `SQLDriverConnect` instead of `SQLConnect` could cause later operations to fail. (Bug#7912)
- An incorrect transaction isolation level may not be returned when accessing the connection attributes. (Bug#27589)
- When inserting data using bulk statements (through `SQLBulkOperations`), the indicators for all rows within the insert would not be updated correctly. (Bug#24306)
- Using `SQLProcedures` does not return the database name within the returned resultset. (Bug#23033)
- Connector/ODBC would return too many foreign key results when accessing tables with similar names. (Bug#4518)
- When using blobs and parameter replacement in a statement with `WHERE CURSOR OF`, the SQL is truncated. (Bug#5853)
- The `SQLTransact()` function did not support an empty connection handle. (Bug#21588)

C.2.17. Changes in Connector/ODBC 3.51.14 (08 March 2007)

Functionality added or changed:

- Use of `SQL_ATTR_CONNECTION_TIMEOUT` on the server has now been disabled. If you attempt to set this attribute on your connection the `SQL_SUCCESS_WITH_INFO` will be returned, with an error number/string of `HYC00: Optional feature not supported`. (Bug#19823)
- Added support for the `HENV` handlers in `SQLEndTran()`.
- Added auto-reconnect option to Connector/ODBC option parameters.
- Added auto is null option to Connector/ODBC option parameters. (Bug#10910)

Bugs fixed:

- The ODBC driver name and version number were incorrectly reported by the driver. (Bug#19740)
- A string format exception would be raised when using iODBC, Connector/ODBC and the embedded MySQL server. (Bug#16535)
- When retrieving data from columns that have been compressed using `COMPRESS()`, the retrieved data would be truncated to 8KB. (Bug#20208)
- When retrieving `TIME` columns, C/ODBC would incorrectly interpret the type of the string and could interpret it as a `DATE` type instead. (Bug#25846)
- On 64-bit systems, some types would be incorrectly returned. (Bug#26024)
- Using `DataAdapter`, Connector/ODBC may continually consume memory when reading the same records within a loop (Windows Server 2003 SP1/SP2 only). (Bug#20459)
- Connector/ODBC may insert the wrong parameter values when using prepared statements under 64-bit Linux. (Bug#22446)
- Using Connector/ODBC, with `SQLBindCol` and binding the length to the return value from `SQL_LEN_DATA_AT_EXEC` fails with a memory allocation error. (Bug#20547)
- The `SQLDriverConnect()` ODBC method did not work with recent Connector/ODBC releases. (Bug#12393)

C.2.18. Changes in Connector/ODBC 3.51.13 (Never released)

Connector/ODBC 3.51.13 was an internal implementation and testing release.

C.2.19. Changes in Connector/ODBC 3.51.12

Functionality added or changed:

- N/A

Bugs fixed:

- Using stored procedures with ADO, where the `CommantType` has been set correctly to `adCmdStoredProc`, calls to stored procedures would fail. ([Bug#15635](#))
- File DSNs could not be saved. ([Bug#12019](#))
- `SQLColumns()` returned no information for tables that had a column named using a reserved word. ([Bug#9539](#))

C.2.20. Changes in Connector/ODBC 3.51.11

Functionality added or changed: No changes.

Bugs fixed:

- `mysql_list_dbcolumns()` and `insert_fields()` were retrieving all rows from a table. Fixed the queries generated by these functions to return no rows. ([Bug#8198](#))
- `SQLGetTypeInfo()` returned `tinyblob` for `SQL_VARBINARY` and nothing for `SQL_BINARY`. Fixed to return `varbinary` for `SQL_VARBINARY`, `binary` for `SQL_BINARY`, and `longblob` for `SQL_LONGVARBINARY`. ([Bug#8138](#))

C.3. Connector/NET Change History

C.3.1. Changes in MySQL Connector/NET Version 5.1.3 (Not yet released)

This is a new Beta development release, fixing recently discovered bugs.

Bugs fixed:

- Extracting data through XML functions within a query returns the data as `System.Byte[]`. This was due to Connector/NET incorrectly identifying `BLOB` fields as binary, rather than text. ([Bug#30233](#))
- Using `TransactionScope` would cause an `InvalidOperationException`. ([Bug#28709](#))
- Connecting to a MySQL server earlier than version 4.1 would raise a `NullException`. ([Bug#29476](#))
- The Saudi Hijri calendar was not supported. ([Bug#29931](#))
- An incorrect `ConstraintException` could be raised on an `INSERT` when adding rows to a table with a multiple-column unique key index. ([Bug#30204](#))
- The availability of a MySQL server would not be reset when using pooled connections (`pooling=true`). This would lead to the server being reported as unavailable, even if the server become available while the application was still running. ([Bug#29409](#))
- A `FormatException` error would be raised if a parameter had not been found, instead of `Re-sources.ParameterMustBeDefined`. ([Bug#29312](#))
- A `DATE` field would be updated with a date/time value, causing a `MySqlDataAdapter.Update()` exception. ([Bug#30077](#))

- Calling `SHOW CREATE PROCEDURE` for routines with a hyphen in the catalog name produced a syntax error. ([Bug#29526](#))
- Using the same connection string multiple times would result in `Database=dbname` appearing multiple times in the resulting string. ([Bug#29123](#))
- Certain operations would not check the `UsageAdvisor` setting, causing log messages from the Usage Advisor even when it was disabled. ([Bug#29124](#))
- Using the membership/role providers when `validationKey` or `decryptionKey` parameters are set to `AutoGenerate`, an exception would be raised when accessing the corresponding values. ([Bug#29235](#))
- An exception would be thrown when using the Manage Role functionality within the web administrator to assign a role to a user. ([Bug#29236](#))
- *Visual Studio Plugin*: Adding a new query based on a stored procedure that uses the `SELECT` statement would terminate the query/TableAdapter wizard. ([Bug#29098](#))

C.3.2. Changes in MySQL Connector/NET Version 5.1.2 (18 June 2007)

This is a new Beta development release, fixing recently discovered bugs.

Note

Connector/NET 5.1.2 now includes the Visual Studio Plugin by default.

Bugs fixed:

- *Visual Studio Plugin*: Update commands would not be generated correctly when using the TableAdapter wizard. ([Bug#26347](#))
- *Visual Studio Plugin*: Adding a new query based on a stored procedure that used a `UPDATE`, `INSERT` or `DELETE` statement would terminate the query/TableAdapter wizard. ([Bug#28536](#))
- *Visual Studio Plugin*: Query Builder would fail to show `TINYTEXT` columns, and any columns listed after a `TINYTEXT` column correctly. ([Bug#28437](#))
- Log messages would be truncated to 300 bytes. ([Bug#28706](#))
- Creating a user would fail due to the application name being set incorrectly. ([Bug#28648](#))
- Accessing the results from a large query when using data compression in the connection would fail to return all the data. ([Bug#28204](#))

C.3.3. Changes in MySQL Connector/NET Version 5.1.1 (23 May 2007)

This is a new Beta development release, fixing recently discovered bugs.

Bugs fixed:

- Running the statement `SHOW PROCESSLIST` would return columns as byte arrays instead of native columns. ([Bug#28448](#))
- Connector/NET would look for the wrong table when executing `User.IsRole()`. ([Bug#28251](#))
- `DATETIME` fields from versions of MySQL before 4.1 would be incorrectly parsed, resulting in an exception. ([Bug#23342](#))
- Installation of the Connector/NET on Windows would fail if VisualStudio had not already been installed. ([Bug#28260](#))
- Using `MySQLDataAdapter.FillSchema()` on a stored procedure would raise an exception: `Invalid attempt to access a field before calling Read()`. ([Bug#27668](#))
- Building a connection string within a tight loop would show slow performance. ([Bug#28167](#))

- The **UNSIGNED** flag for parameters in a stored procedure would be ignored when using `MySQLCommandBuilder` to obtain the parameter information. ([Bug#27679](#))
- Fixed password property on `MySQLConnectionStringBuilder` to use `PasswordPropertyText` attribute. This causes dots to show instead of actual password text.

C.3.4. Changes in MySQL Connector/NET Version 5.1.0 (01 May 2007)

Functionality added or changed:

This is the first public alpha release of the current Connector/NET 5.1 development branch, providing an insight to upcoming features. Although some of these are still under development, this release includes the following new features and changes (in comparison to the current Connector/NET 5.0 production release):

- Added Membership and Role provider contributed by Sean Wright (thanks!).
- Now compiles for .NET CF 2.0.
- Rewrote stored procedure parsing code using a new SQL tokenizer. Really nasty procedures including nested comments are now supported.
- GetSchema will now report objects relative to the currently selected database. What this means is that passing in null as a database restriction will report objects on the currently selected database only.

C.3.5. Changes in MySQL Connector/NET Version 5.0.8 (21 August 2007)

Note

This version introduces a new installer technology.

Bugs fixed:

- Extracting data through XML functions within a query returns the data as `System.Byte[]`. This was due to Connector/NET incorrectly identifying **BLOB** fields as binary, rather than text. ([Bug#30233](#))
- Fixed problem where we were not closing prepared statement handles when commands are disposed. This could lead to using up all prepared statement handles on the server.
- Fixed problem where any attempt to not read all the records returned from a select where each row of the select is greater than 1024 bytes would hang the driver.
- Fixed problem where a command timing out just after it actually finished would cause an exception to be thrown on the command timeout thread which would then be seen as an unhandled exception.
- Fixed bug where Connector/Net was hand building some date time patterns rather than using the patterns provided under Culture-Info. This caused problems with some calendars that do not support the same ranges as Gregorian.. ([Bug#29931](#))
- Fixed problem where `MySQLConnection.BeginTransaction` checked the drivers status var before checking if the connection was open. The result was that the driver could report an invalid condition on a previously opened connection.
- Fixed the database schema collection so that it works on servers that are not properly respecting the `lower_case_table_names` setting.
- Fixed some serious issues with command timeout and cancel that could present as exceptions about thread ownership. The issue was that not all queries cancel the same. Some produce resultsets while others don't. `ExecuteReader` had to be changed to check for this.
- An incorrect `ConstraintException` could be raised on an **INSERT** when adding rows to a table with a multiple-column unique key index. ([Bug#30204](#))
- The availability of a MySQL server would not be reset when using pooled connections (`pooling=true`). This would lead to the server being reported as unavailable, even if the server become available while the application was still running. ([Bug#29409](#))

- A `FormatException` error would be raised if a parameter had not been found, instead of `Re-sources.ParameterMustBeDefined`. (Bug#29312)
- A `DATE` field would be updated with a date/time value, causing a `MySqlDataAdapter.Update()` exception. (Bug#30077)
- Calling `SHOW CREATE PROCEDURE` for routines with a hyphen in the catalog name produced a syntax error. (Bug#29526)
- Using the same connection string multiple times would result in `Database=dbname` appearing multiple times in the resulting string. (Bug#29123)
- Certain operations would not check the `UsageAdvisor` setting, causing log messages from the Usage Advisor even when it was disabled. (Bug#29124)
- Log messages would be truncated to 300 bytes. (Bug#28706)
- Accessing the results from a large query when using data compression in the connection will fail to return all the data. (Bug#28204)

C.3.6. Changes in MySQL Connector/NET Version 5.0.7 (18 May 2007)

Bugs fixed:

- Running the statement `SHOW PROCESSLIST` would return columns as byte arrays instead of native columns. (Bug#28448)
- Enlisting a null transaction would affect the current connection object, such that further enlistment operations to the transaction are not possible. (Bug#26754)
- Attempting to change the `Connection Protocol` property within a `PropertyGrid` control would raise an exception. (Bug#26472)
- `DATETIME` fields from versions of MySQL before 4.1 would be incorrectly parsed, resulting in an exception. (Bug#23342)
- Using `MySqlDataAdapter.FillSchema()` on a stored procedure would raise an exception: `Invalid attempt to access a field before calling Read()`. (Bug#27668)
- Building a connection string within a tight loop would show slow performance. (Bug#28167)
- The `UNSIGNED` flag for parameters in a stored procedure would be ignored when using `MySqlCommandBuilder` to obtain the parameter information. (Bug#27679)
- Using logging (with the `logging=true` parameter to the connection string) would not generate a log file. (Bug#27765)
- The `CreateFormat` column of the `DataTypes` collection did not contain a format specification for creating a new column type. (Bug#25947)
- The `characterset` property would not be identified during a connection (also affected Visual Studio Plugin). (Bug#27240, Bug#26147)
- When cloning an open `MySqlClient.MySqlConnection` with the `Persist Security Info=False` option set, the cloned connection is not usable because the security information has not been cloned. (Bug#27269)
- If you close an open connection with an active transaction, the transaction is not automatically rolled back. (Bug#27289)

C.3.7. Changes in MySQL Connector/NET Version 5.0.6 (22 March 2007)

Bugs fixed:

- `MySqlParameterCollection` and parameters added with `Insert` method can not be retrieved later using `ParameterName`. (Bug#27135)
- Publisher listed in "Add/Remove Programs" is not consistent with other MySQL products. (Bug#27253)

- `cmd.Parameters.RemoveAt("Id")` will cause an error if the last item is requested. (Bug#27187)
- Exception thrown when using large values in `UInt64` parameters. (Bug#27093)
- `DESCRIBE . . . .` SQL command returns byte arrays rather than data on MySQL versions older than 4.1.15. (Bug#27221)
- MySQL Visual Studio Plugin 1.1.2 does not work with Connector/Net 5.0.5. (Bug#26960)

C.3.8. Changes in MySQL Connector/NET Version 5.0.5 (07 March 2007)

Functionality added or changed:

- Return parameters created with `DeriveParameters` now have the name `RETURN_VALUE`.
- Fixed problem with parameter name hashing where the hashes were not getting updated when parameters were removed from the collection.
- Fixed problem with calling stored functions when a return parameter was not given.
- Fixed problem that prevented use of `SchemaOnly` or `SingleRow` command behaviors with stored procedures or prepared statements.
- Assembly now properly appears in the Visual Studio 2005 Add/Remove Reference dialog.
- Added `MySqlParameterCollection.AddWithValue` and marked the `Add(name, value)` method as obsolete.
- Added `Use Procedure Bodies` connection string option to allow calling procedures without using procedure metadata.
- Reverted behavior that required parameter names to start with the parameter marker. We apologize for this back and forth but we mistakenly changed the behavior to not match what `SqlClient` supports. We now support using either syntax for adding parameters however we also respond exactly like `SqlClient` in that if you ask for the index of a parameter using a syntax different from when you added the parameter, the result will be -1.

Bugs fixed:

- The `UpdateRowSource.FirstReturnedRecord` method does not work. (Bug#25569)
- A critical `ConnectionPool` error would result in repeated `System.NullReferenceException`. (Bug#25603)
- Applications would crash when calling with `CommandType` set to `StoredProcedure`.
- `BINARY` and `VARBINARY` columns would be returned as a string, not binary, datatype. (Bug#25605)
- `MySqlConnection.GetSchema` fails with `NullReferenceException` for Foreign Keys. (Bug#26660)
- When a `MySqlConversionException` is raised on a remote object, the client application would receive a `SerializationException` instead. (Bug#24957)
- High CPU utilization would be experienced when there is no idle connection waiting when using pooled connections through `MySqlPool.GetConnection`. (Bug#24373)
- Connector/NET would not compile properly when used with Mono 1.2. (Bug#24263)
- Opening a connection would be slow due to hostname lookup. (Bug#26152).
- Connector/NET would fail to install under Windows Vista. (Bug#26430)
- Incorrect values/formats would be applied when the `OldSyntax` connection string option was used. (Bug#25950)
- Registry would be incorrectly populated with installation locations. (Bug#25928)
- Filling a table schema through a stored procedure triggers a runtime error. (Bug#25609)

- `MySqlConnection` throws an exception when connecting to MySQL v4.1.7. ([Bug#25726](#))
- Returned data types of a `DataTypes` collection do not contain the right correct CLR Datatype. ([Bug#25907](#))
- `GetSchema` and `DataTypes` would throw an exception due to an incorrect table name. ([Bug#25906](#))
- Times with negative values would be returned incorrectly. ([Bug#25912](#))
- `SELECT` did not work correctly when using a `WHERE` clause containing a UTF-8 string. ([Bug#25651](#))
- When closing and then re-opening a connection to a database, the character set specification is lost. ([Bug#25614](#))
- When connecting to a MySQL Server earlier than version 4.1, the connection would hang when reading data. ([Bug#25458](#))
- When connecting to a server, the return code from the connection could be zero, even though the hostname was incorrect. ([Bug#24802](#))
- Using `ExecuteScalar()` with more than one query, where one query fails, will hang the connection. ([Bug#25443](#))

C.3.9. Changes in MySQL Connector/NET Version 5.0.4 (Not released)

Connector/NET Version 5.0.4 was not released due to an incomplete bug fix.

C.3.10. Changes in MySQL Connector/NET Version 5.0.3 (05 January 2007)

Functionality added or changed:

- SSL support has been updated.
- The `ViewColumns GetSchema` collection has been updated.
- The `CommandBuilder.DeriveParameters` function has been updated to the procedure cache.
- Improved speed and performance by re-architecting certain sections of the code.
- Usage Advisor has been implemented. The Usage Advisor checks your queries and will report if you are using the connection inefficiently.
- The `MySQLCommand` object now supports asynchronous query methods. This is implemented using the `BeginExecuteNonQuery` and `EndExecuteNonQuery` methods.
- Metadata from stored procedures and stored function execution are cached.
- PerfMon hooks have been added to monitor the stored procedure cache hits and misses.
- The `ShapZipLib` library has been replaced with the deflate support provided within .NET 2.0.
- Support for the embedded server and client library have been removed from this release. Support will be added back to a later release.

Bugs fixed:

- An exception would be raised, or the process would hang, if `SELECT` privileges on a database were not granted and a stored procedure was used. ([Bug#25033](#))
- Nested transactions (which are unsupported) do not raise an error or warning. ([Bug#22400](#))
- When adding parameter objects to a command object, if the parameter direction is set to `ReturnValue` before the parameter is added to the command object then when the command is executed it throws an error. ([Bug#25013](#))
- Additional text added to error message ([Bug#25178](#))

- Stored procedure executions are not thread safe. ([Bug#23905](#))
- Using `Driver.IsTooOld()` would return the wrong value. ([Bug#24661](#))
- Deleting a connection to a disconnected server when using the Visual Studio Plugin would cause an assertion failure. ([Bug#23687](#))
- When using a `DBNull.Value` as the value for a parameter value, and then later setting a specific value type, the command would fail with an exception because the wrong type was implied from the `DBNull.Value`. ([Bug#24565](#))

C.3.11. Changes in MySQL Connector/NET Version 5.0.2 (06 November 2006)

Functionality added or changed:

- **Important change:** Due to a number of issues with the use of server-side prepared statements, Connector/NET 5.0.2 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string properties:

```
ignore prepare=false
```

The default value of this property is true.

- Implemented a stored procedure cache. By default, the connector caches the metadata for the last 25 procedures that are seen. You can change the number of procedures that are cached by using the `procedure cache` connection string.
- An `Ignore Prepare` option has been added to the connection string options. If enabled, prepared statements will be disabled application-wide. The default for this option is true.

Bugs fixed:

- Creating a connection through the Server Explorer when using the Visual Studio Plugin would fail. The installer for the Visual Studio Plugin has been updated to ensure that Connector/NET 5.0.2 must be installed. ([Bug#23071](#))
- Within Mono, using the `PreparedStatement` interface could result in an error due to a `BitArray` copying error. ([Bug#18186](#))
- Using Windows Vista (RC2) as a non-privileged user would raise a `Registry key 'Global' access denied`. ([Bug#22882](#))
- One system where IPv6 was enabled, Connector/NET would incorrectly resolve hostnames. ([Bug#23758](#))
- Column names with accented characters were not parsed properly causing malformed column names in result sets. ([Bug#23657](#))
- Connector/NET did not work as a data source for the `SqlDataSource` object used by ASP.NET 2.0. ([Bug#16126](#))
- A `System.FormatException` exception would be raised when invoking a stored procedure with an `ENUM` input parameter. ([Bug#23268](#))
- During installation, an antivirus error message would be raised (indicating a malicious script problem). ([Bug#23245](#))
- An exception would be thrown when calling `GetSchemaTable` and `fields` was null. ([Bug#23538](#))

C.3.12. Changes in MySQL Connector/NET Version 5.0.1 (01 October 2006)

Bugs fixed:

- Using `ExecuteScalar` with a datetime field, where the value of the field is "0000-00-00 00:00:00", a `MySQLConversionException` exception would be raised. ([Bug#11991](#))
- An `MySQL.Data.Types.MySQLConversionException` would be raised when trying to update a row that contained a date

field, where the date field contained a zero value (0000-00-00 00:00:00). (Bug#9619)

- Incorrect field/data lengths could be returned for `VARCHAR` UTF8 columns. Bug (#14592)
- Submitting an empty string to a command object through `prepare` raises an `System.IndexOutOfRangeException`, rather than a Connector/Net exception. (Bug#18391)
- Starting a transaction on a connection created by `MySql.Data.MySqlClient.MySqlClientFactory`, using `BeginTransaction` without specifying an isolation level, causes the SQL statement to fail with a syntax error. (Bug#22042)
- Connector/NET on a Turkish operating system, may fail to execute certain SQL statements correctly. (Bug#22452)
- You can now install the Connector/NET MSI package from the command line using the `/passive`, `/quiet`, `/q` options. (Bug#19994)
- The `MySqlException` class is now derived from the `DbException` class. (Bug#21874)
- Executing multiple queries as part of a transaction returns `There is already an open DataReader associated with this Connection which must be closed first`. (Bug#7248)
- The `#` would not be accepted within column/table names, even though it was valid. (Bug#21521)

C.3.13. Changes in MySQL Connector/NET Version 5.0.0 (08 August 2006)

This is a new Alpha development release, fixing recently discovered bugs.

NOTE: This Alpha release, as any other pre-production release, should not be installed on *production* level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

Bugs fixed:

- `CommandText`: Question mark in comment line is being parsed as a parameter. (Bug#6214)

Functionality added or changed:

- Implemented Usage Advisor.
- Added Async query methods.
- Reimplemented `PacketReader/PacketWriter` support into `MySqlStream` class.
- Added internal implementation of SHA1 so we don't have to distribute the OpenNetCF on mobile devices.
- Added usage advisor warnings for requesting column values by the wrong type.
- Reworked connection string classes to be simpler and faster.
- Added procedure metadata caching.
- Added perfmon hooks for stored procedure cache hits and misses.
- Implemented `MySqlConnectionBuilder` class.
- Implemented `MySqlClientFactory` class.
- Implemented classes and interfaces for ADO.Net 2.0 support.
- Replaced use of `ICSharpCode` with .NET 2.0 internal deflate support.
- Refactored test suite to test all protocols in a single pass.

- Completely refactored how column values are handled to avoid boxing in some cases.

C.3.14. Changes in MySQL Connector/NET Version 1.0.11 (Not yet released)

Bugs fixed:

- Extracting data through XML functions within a query returns the data as `System.Byte[]`. This was due to Connector/NET incorrectly identifying `BLOB` fields as binary, rather than text. ([Bug#30233](#))

C.3.15. Changes in MySQL Connector/NET Version 1.0.10 (24 August 2007)

Bugs fixed:

- The availability of a MySQL server would not be reset when using pooled connections (`pooling=true`). This would lead to the server being reported as unavailable, even if the server become available while the application was still running. ([Bug#29409](#))
- An incorrect `ConstraintException` could be raised on an `INSERT` when adding rows to a table with a multiple-column unique key index. ([Bug#30204](#))
- Publisher listed in "Add/Remove Programs" is not consistent with other MySQL products. ([Bug#27253](#))
- `MySQLParameterCollection` and parameters added with `Insert` method can not be retrieved later using `ParameterName`. ([Bug#27135](#))
- A critical `ConnectionPool` error would result in repeated `System.NullReferenceException`. ([Bug#25603](#))
- When a `MySQLConversionException` is raised on a remote object, the client application would receive a `SerializationException` instead. ([Bug#24957](#))
- High CPU utilization would be experienced when there is no idle connection waiting when using pooled connections through `MySQLPool.GetConnection`. ([Bug#24373](#))
- `BINARY` and `VARBINARY` columns would be returned as a string, not binary, datatype. ([Bug#25605](#))

C.3.16. Changes in MySQL Connector/NET Version 1.0.9 (02 February 2007)

Functionality added or changed:

- **Important change:** Due to a number of issues with the use of server-side prepared statements, Connector/NET 5.0.2 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string properties:

```
ignore prepare=false
```

The default value of this property is true.

- **Important change:** Binaries for .NET 1.0 are no longer supplied with this release. If you need support for .NET 1.0, you must build from source.
- Implemented a stored procedure cache. By default, the connector caches the metadata for the last 25 procedures that are seen. You can change the number of procedures that are cached by using the `procedure cache` connection string.
- An `Ignore Prepare` option has been added to the connection string options. If enabled, prepared statements will be disabled application-wide. The default for this option is true.
- The ICSharpCode ZipLib is no longer used by the Connector, and is no longer distributed with it.

- Improved `CommandBuilder.DeriveParameters` to first try and use the procedure cache before querying for the stored procedure metadata. Return parameters created with `DeriveParameters` now have the name `RETURN_VALUE`.

Bugs fixed:

- Trying to fill a table schema through a stored procedure triggers a runtime error. ([Bug#25609](#))
- `MySQLConnection` throws a `NullReferenceException` and `ArgumentNullException` when connecting to MySQL v4.1.7. ([Bug#25726](#))
- `SELECT` did not work correctly when using a `WHERE` clause containing a UTF-8 string. ([Bug#25651](#))
- Times with negative values would be returned incorrectly. ([Bug#25912](#))
- When closing and then re-opening a connection to a database, the character set specification is lost. ([Bug#25614](#))
- When connecting to a server, the return code from the connection could be zero, even though the hostname was incorrect. ([Bug#24802](#))
- Using `ExecuteScalar()` with more than one query, where one query fails, will hang the connection. ([Bug#25443](#))
- Nested transactions do not raise an error or warning. ([Bug#22400](#))
- When adding parameter objects to a command object, if the parameter direction is set to `ReturnValue` before the parameter is added to the command object then when the command is executed it throws an error. ([Bug#25013](#))
- Additional text added to error message. ([Bug#25178](#))
- The `CommandBuilder` would mistakenly add insert parameters for a table column with auto incrementation enabled. ([Bug#23862](#))
- Stored procedure executions are not thread safe. ([Bug#23905](#))
- Using `Driver.IsTooOld()` would return the wrong value. ([Bug#24661](#))
- When using a `DBNull.Value` as the value for a parameter value, and then later setting a specific value type, the command would fail with an exception because the wrong type was implied from the `DBNull.Value`. ([Bug#24565](#))
- Within Mono, using the `PreparedStatement` interface could result in an error due to a `BitArray` copying error. (Bug 18186)
- One system where IPv6 was enabled, Connector/NET would incorrectly resolve hostnames. ([Bug#23758](#))
- An `System.OverflowException` would be raised when accessing a varchar field over 255 bytes. Bug (#23749)

C.3.17. Changes in MySQL Connector/NET Version 1.0.8 (20 October 2006)

Bugs fixed:

- Using `ExecuteScalar` with a datetime field, where the value of the field is "0000-00-00 00:00:00", a `MySQLConversionException` exception would be raised. ([Bug#11991](#))
- The `MySQLDateTime` class did not contain constructors. ([Bug#15112](#))
- `DataReader` would show the value of the previous row (or last row with non-null data) if the current row contained a `datetime` field with a null value. ([Bug#16884](#))
- An `MySQL.Data.Types.MySQLConversionException` would be raised when trying to update a row that contained a date field, where the date field contained a zero value (0000-00-00 00:00:00). ([Bug#9619](#))
- When using `MySQLDataAdapter`, connections to a MySQL server may remain open and active, even though the use of the connection has been completed and the data received. ([Bug#8131](#))

- Incorrect field/data lengths could be returned for `VARCHAR` UTF8 columns. Bug (#14592)
- Submitting an empty string to a command object through `prepare` raises an `System.IndexOutOfRangeException`, rather than a Connector/Net exception. (Bug#18391)
- Connector/NET on a Turkish operating system, may fail to execute certain SQL statements correctly. (Bug#22452)
- You can now install the Connector/NET MSI package from the command line using the `/passive`, `/quiet`, `/q` options. (Bug#19994)
- Executing multiple queries as part of a transaction returns `There is already an openDataReader associated with this Connection which must be closed first`. (Bug#7248)
- Calling `MySqlCommandBuilder.DeriveParameters` for a stored procedure that has no parameters would cause an application crash. (Bug#15077)
- A `SELECT` query on a table with a date with a value of `'0000-00-00'` would hang the application. (Bug#17736)
- The `#` would not be accepted within column/table names, even though it was valid. (Bug#21521)
- Calling `Close` on a connection after calling a stored procedure would trigger a `NullReferenceException`. (Bug#20581)
- `IDataRecord.GetString` would raise `NullPointerException` for null values in returned rows. Method now throws `SqlNullValueException`. (Bug#19294)
- An exception would be raised when using an output parameter to a `System.String` value. (Bug#17814)
- The `DiscoverParameters` function would fail when a stored procedure used a `NUMERIC` parameter type. (Bug#19515)
- When running a query that included a date comparison, a `DateReader` error would be raised. (Bug#19481)
- Parameter substitution in queries where the order of parameters and table fields did not match would substitute incorrect values. (Bug#19261)
- When working with multiple threads, character set initialization would generate errors. (Bug#17106)
- When using an unsigned 64-bit integer in a stored procedure, the unsigned bit would be lost stored. (Bug#16934)
- The connection string parser did not allow single or double quotes in the password. (Bug#16659)
- The `CommandBuilder` ignored `Unsigned` flag at `Parameter` creation. (Bug#17375)
- `CHAR` type added to `MySqlDbType`. (Bug#17749)
- Unsigned data types were not properly supported. (Bug#16788)

Functionality added or changed:

- Stored procedures are now cached.
- The method for retrieving stored procedure metadata has been changed so that users without `SELECT` privileges on the `mysql.proc` table can use a stored procedure.

C.3.18. Changes in MySQL Connector/NET Version 1.0.7 (21 November 2005)

- Unsigned `tinyint` (NET byte) would lead to and incorrectly determined parameter type from the parameter value. (Bug#18570)
- The parameter collection object's `Add()` method added parameters to the list without first checking to see whether they already existed. Now it updates the value of the existing parameter object if it exists. (Bug#13927)
- A `#42000Query was empty` exception occurred when executing a query built with `MySqlCommandBuilder`, if the query string ended with a semicolon. (Bug#14631)

- Implemented the `MySqlCommandBuilder.DeriveParameters` method that is used to discover the parameters for a stored procedure. ([Bug#13632](#))
- Added support for the `cp932` character set. ([Bug#13806](#))
- Calling a stored procedure where a parameter contained special characters (such as '@') would produce an exception. Note that `ANSI_QUOTES` had to be enabled to make this possible. ([Bug#13753](#))
- A statement that contained multiple references to the same parameter could not be prepared. ([Bug#13541](#))
- The `Ping()` method did not update the `State` property of the `Connection` object. ([Bug#13658](#))

C.3.19. Changes in MySQL Connector/NET Version 1.0.6 (03 October 2005)

- The `nant` build sequence had problems. ([Bug#12978](#))
- Serializing a parameter failed if the first value passed in was `NULL`. ([Bug#13276](#))
- Field names that contained the following characters caused errors: `( )%<>/` ([Bug#13036](#))
- The Connector/NET 1.0.5 installer would not install alongside Connector/NET 1.0.4. ([Bug#12835](#))
- Connector/NET 1.0.5 could not connect on Mono. ([Bug#13345](#))

C.3.20. Changes in MySQL Connector/NET Version 1.0.5 (29 August 2005)

- With multiple hosts in the connection string, Connector/NET would not connect to the last host in the list. ([Bug#12628](#))
- Connector/NET interpreted the new decimal data type as a byte array. ([Bug#11294](#))
- The `cp1250` character set was not supported. ([Bug#11621](#))
- Connection could fail when .NET thread pool had no available worker threads. ([Bug#10637](#))
- Decimal parameters caused syntax errors. ([Bug#11550](#), [Bug#10486](#), [Bug#10152](#))
- A call to a stored procedure caused an exception if the stored procedure had no parameters. ([Bug#11542](#))
- Certain malformed queries would trigger a `Connection must be valid and open` error message. ([Bug#11490](#))
- The `MySqlCommandBuilder` class could not handle queries that referenced tables in a database other than the default database. ([Bug#8382](#))
- Connector/NET could not work properly with certain regional settings. (WL#8228)
- Trying to use a stored procedure when `Connection.Database` was not populated generated an exception. ([Bug#11450](#))
- Trying to read a `TIMESTAMP` column generated an exception. ([Bug#7951](#))
- Parameters were not recognized when they were separated by linefeeds. ([Bug#9722](#))
- Calling `MySqlConnection.clone` when a connection string had not yet been set on the original connection would generate an error. ([Bug#10281](#))
- Added support to call a stored function from Connector/NET. ([Bug#10644](#))
- Connector/NET could not connect to MySQL 4.1.14. ([Bug#12771](#))
- The `ConnectionString` property could not be set when a `MySqlConnection` object was added with the designer. ([Bug#12551](#), [Bug#8724](#))

C.3.21. Changes in MySQL Connector/NET Version 1.0.4 (20 January 2005)

- Calling prepare causing exception. ([Bug#7243](#))
- Fixed another small problem with prepared statements.
- `MySqlCommand.Connection` returns an `IDbConnection`. ([Bug#7258](#))
- `MySqlAdapter.Fill` method throws error message `Non-negative number required`. ([Bug#7345](#))
- Clone method bug in `MySqlCommand`. ([Bug#7478](#))
- `MySqlDataReader.GetString(index)` returns non-Null value when field is `Null`. ([Bug#7612](#))
- `MySqlReader.GetInt32` throws exception if column is unsigned. ([Bug#7755](#))
- `GetBytes` is working no more. ([Bug#7704](#)).
- Quote character `\222` not quoted in `EscapeString`. ([Bug#7724](#))
- Fixed problem that causes named pipes to not work with some blob functionality.
- Fixed problem with shared memory connections.
- Problem with Multiple resultsets. ([Bug#7436](#))
- Added or filled out several more topics in the API reference documentation.

C.3.22. Changes in MySQL Connector/NET Version 1.0.3-gamma (12 October 2004)

- Made MySQL the default named pipe name.
- Now `SHOW COLLATION` is used upon connection to retrieve the full list of charset ids.
- Fixed Invalid character set index: 200. ([Bug#6547](#))
- Installer now includes options to install into GAC and create `START MENU` items.
- Int64 Support in `MySqlCommand` Parameters. ([Bug#6863](#))
- Connections now do not have to give a database on the connection string.
- `MySqlDataReader.GetChar(int i)` throws `IndexOutOfRangeException` exception. ([Bug#6770](#))
- Fixed problem where multiple resultsets having different numbers of columns would cause a problem.
- Exception stack trace lost when re-throwing exceptions. ([Bug#6983](#))
- Fixed major problem with detecting null values when using prepared statements.
- Errors in parsing stored procedure parameters. ([Bug#6902](#))
- Integer "out" parameter from stored procedure returned as string. ([Bug#6668](#))
- `MySqlDateTime` in Databases sorting by Text, not Date. ([Bug#7032](#))
- Invalid query string when using inout parameters ([Bug#7133](#))
- Test suite fails with MySQL 4.0 because of case sensitivity of table names. ([Bug#6831](#))
- Inserting `DateTime` causes `System.InvalidCastException` to be thrown. ([Bug#7132](#))

- InvalidCast when using `DATE_ADD`-function. ([Bug#6879](#))
- An Open Connection has been Closed by the Host System. ([Bug#6634](#))
- Added `ServerThread` property to `MySqlConnection` to expose server thread id.
- Added Ping method to `MySqlConnection`.
- Changed the name of the test suite to `MySQL.Data.Tests.dll`.

C.3.23. Changes in MySQL Connector/NET Version 1.0.2-gamma (15 November 2004)

- Fixed problem with `MySqlBinary` where string values could not be used to update extended text columns
- Fixed Installation directory ignored using custom installation ([Bug#6329](#))
- Fixed problem where setting command text leaves the command in a prepared state
- Fixed double type handling in `MySqlParameter(string parameterName, object value)` ([Bug#6428](#))
- Fixed Zero date "0000-00-00" is returned wrong when filling Dataset ([Bug#6429](#))
- Fixed problem where calling stored procedures might cause an "Illegal mix of collations" problem.
- Added charset connection string option
- Fixed #HY000 Illegal mix of collations (latin1_swedish_ci,IMPLICIT) and (utf8_general_ ([Bug#6322](#))
- Added the TableEditor CS and VB sample
- Fixed Charset-map for UCS-2 ([Bug#6541](#))
- Updated the installer to include the new samples
- Fixed Long inserts take very long time (Bu #5453)
- Fixed Objects not being disposed ([Bug#6649](#))
- Provider is now using character set specified by server as default

C.3.24. Changes in MySQL Connector/NET Version 1.0.1-beta2 (27 October 2004)

- Fixed [Bug#5602](#) Possible bug in `MySqlParameter(string, object)` constructor
- Fixed [Bug#5458](#) Calling `GetChars` on a longtext column throws an exception
- Fixed [Bug#5474](#) cannot run a stored procedure populating `mysqlcommand.parameters`
- Fixed [Bug#5469](#) Setting `DbType` throws `NullReferenceException`
- Fixed problem where connector was not issuing a `CMD_QUIT` before closing the socket
- Fixed [Bug#5392](#) `MySqlCommand` sees "?" as parameters in string literals
- Fixed problem with `ConnectionInternal` where a key might be added more than once
- CP1252 is now used for Latin1 only when the server is 4.1.2 and later
- Fixed [Bug#5388](#) `DataReader` reports all rows as NULL if one row is NULL

- Virtualized driver subsystem so future releases could easily support client or embedded server support
- Field buffers being reused to decrease memory allocations and increase speed
- Fixed problem where using old syntax while using the interfaces caused problems
- Using PacketWriter instead of Packet for writing to streams
- Refactored compression code into CompressedStream to clean up NativeDriver
- Added test case for resetting the command text on a prepared command
- Fixed problem where MySqlParameterCollection.Add() would throw unclear exception when given a null value ([Bug#5621](#))
- Fixed constructor initialize problems in MySqlCommand() ([Bug#5613](#))
- Fixed Parsing the ';' char ([Bug#5876](#))
- Fixed missing Reference in DbType setter ([Bug#5897](#))
- Fixed System.OverflowException when using YEAR datatype ([Bug#6036](#))
- Added Aggregate function test (wasn't really a bug)
- Fixed serializing of floating point parameters (double, numeric, single, decimal) ([Bug#5900](#))
- IsNullable error ([Bug#5796](#))
- Fixed problem where connection lifetime on the connect string was not being respected
- Fixed problem where Min Pool Size was not being respected
- Fixed MySqlDataReader and 'show tables from ...' behavior ([Bug#5256](#))
- Implemented SequentialAccess
- Fixed MySqlDateTime sets IsZero property on all subseq.records after first zero found ([Bug#6006](#))
- Fixed Can't display Chinese correctly ([Bug#5288](#))
- Fixed Russian character support as well
- Fixed Method TokenizeSql() uses only a limited set of valid characters for parameters ([Bug#6217](#))
- Fixed NET Connector source missing resx files ([Bug#6216](#))
- Fixed DBNull Values causing problems with retrieving/updating queries. ([Bug#5798](#))
- Fixed Yet Another "object reference not set to an instance of an object" ([Bug#5496](#))
- Fixed problem in PacketReader where it could try to allocate the wrong buffer size in EnsureCapacity
- Fixed GetBoolean returns wrong values ([Bug#6227](#))
- Fixed IndexOutOfBounds when reading BLOB with DataReader with GetString(index) ([Bug#6230](#))

C.3.25. Changes in MySQL Connector/NET Version 1.0.0 (01 September 2004)

- Thai encoding not correctly supported. ([Bug#3889](#))
- Updated many of the test cases.
- Fixed problem with using compression.

- Bumped version number to 1.0.0 for beta 1 release.
- Added [COPYING.rtf](#) file for use in installer.
- Removed all of the XML comment warnings.
- Removed some last references to ByteFX.

C.3.26. Changes in MySQL Connector/NET Version 0.9.0 (30 August 2004)

- Added test fixture for prepared statements.
- All type classes now implement a [SerializeBinary](#) method for sending their data to a [PacketWriter](#).
- Added [PacketWriter](#) class that will enable future low-memory large object handling.
- Fixed many small bugs in running prepared statements and stored procedures.
- Changed command so that an exception will not be thrown in executing a stored procedure with parameters in old syntax mode.
- [SingleRow](#) behavior now working right even with limit.
- [GetBytes](#) now only works on binary columns.
- Logger now truncates long sql commands so blob columns don't blow out our log.
- host and database now have a default value of "" unless otherwise set.
- Connection Timeout seems to be ignored. ([Bug#5214](#))
- Added test case for bug# 5051: GetSchema not working correctly.
- Fixed problem where [GetSchema](#) would return false for [IsUnique](#) when the column is key.
- [MySqlDataReader](#) [GetXXX](#) methods now using the field level [MySqlValue](#) object and not performing conversions.
- [DataReader](#) returning NULL for time column. ([Bug#5097](#))
- Added test case for [LOAD DATA LOCAL INFILE](#).
- Added replacetext custom nant task.
- Added [CommandBuilderTest](#) fixture.
- Added Last One Wins feature to [CommandBuilder](#).
- Fixed persist security info case problem.
- Fixed [GetBool](#) so that 1, true, "true", and "yes" all count as true.
- Make parameter mark configurable.
- Added the "old syntax" connection string parameter to allow use of @ parameter marker.
- [MySqlCommandBuilder](#). ([Bug#4658](#))
- [ByteFX.MySqlClient](#) caches passwords if [Persist Security Info](#) is false. ([Bug#4864](#))
- Updated license banner in all source files to include FLOSS exception.
- Added new .Types namespace and implementations for most current MySql types.
- Added [MySqlField41](#) as a subclass of [MySqlField](#).

- Changed many classes to now use the new `.Types` types.
- Changed type `enum int` to `Int32`, `short` to `Int16`, and `bigint` to `Int64`.
- Added dummy types `UInt16`, `UInt32`, and `UInt64` to allow an unsigned parameter to be made.
- Connections are now reset when they are pulled from the connection pool.
- Refactored auth code in driver so it can be used for both auth and reset.
- Added `UserReset` test in `PoolingTests.cs`.
- Connections are now reset using `COM_CHANGE_USER` when pulled from the pool.
- Implemented `SingleResultSet` behavior.
- Implemented support of unicode.
- Added char set mappings for utf-8 and ucs-2.
- Time fields overflow using bytefx .net mysql driver ([Bug#4520](#))
- Modified time test in data type test fixture to check for time spans where hours > 24.
- Wrong string with backslash escaping in `ByteFx.Data.MySqlClient.MySqlParameter`. ([Bug#4505](#))
- Added code to Parameter test case TestQuoting to test for backslashes.
- `MySqlCommandBuilder` fails with multi-word column names. ([Bug#4486](#))
- Fixed bug in `TokenizeSql` where underscore would terminate character capture in parameter name.
- Added test case for spaces in column names.
- `MySqlDataReader.GetBytes` don't works correctly. ([Bug#4324](#))
- Added `GetBytes()` test case to `DataReader` test fixture.
- Now reading all server variables in `InternalConnection.Configure` into `Hashtable`.
- Now using `string[]` for index map in `CharSetMap`.
- Added `CRInSQL` test case for carriage returns in SQL.
- Setting `maxPacketSize` to default value in `Driver.ctor`.
- Setting `MySqlDbType` on a parameter doesn't set generic type. ([Bug#4442](#))
- Removed obsolete data types `Long` and `LongLong`.
- Overflow exception thrown when using "use pipe" on connection string. ([Bug#4071](#))
- Changed "use pipe" keyword to "pipe name" or just "pipe".
- Allow reading multiple resultsets from a single query.
- Added flags attribute to `ServerStatusFlags` enum.
- Changed name of `ServerStatus` enum to `ServerStatusFlags`.
- Inserted data row doesn't update properly.
- Error processing show create table. ([Bug#4074](#))
- Change `Packet.ReadLenInteger` to `ReadPackedLong` and added `packet.ReadPackedInteger` that always reads integers packed with 2,3,4.

- Added `syntax.cs` test fixture to test various SQL syntax bugs.
- Improper handling of time values. Now time value of 00:00:00 is not treated as null. (Bug#4149)
- Moved all test suite files into `TestSuite` folder.
- Fixed bug where null column would move the result packet pointer backward.
- Added new nant build script.
- Clear tablename so it will be regen'ed properly during the next `GenerateSchema`. (Bug#3917)
- `GetValues` was always returning zero and was also always trying to copy all fields rather than respecting the size of the array passed in. (Bug#3915)
- Implemented shared memory access protocol.
- Implemented prepared statements for MySQL 4.1.
- Implemented stored procedures for MySQL 5.0.
- Renamed `MySQLInternalConnection` to `InternalConnection`.
- SQL is now parsed as chars, fixes problems with other languages.
- Added logging and allow batch connection string options.
- `RowUpdating` event not set when setting the `DataAdapter` property. (Bug#3888)
- Fixed bug in char set mapping.
- Implemented 4.1 authentication.
- Improved open/auth code in driver.
- Improved how connection bits are set during connection.
- Database name is now passed to server during initial handshake.
- Changed namespace for client to `MySQL.Data.MySqlClient`.
- Changed assembly name of client to `MySQL.Data.dll`.
- Changed license text in all source files to GPL.
- Added the `MySQLClient.build` Nant file.
- Removed the mono batch files.
- Moved some of the unused files into `notused` folder so nant build file can use wildcards.
- Implemented shared memory access.
- Major revamp in code structure.
- Prepared statements now working for MySQL 4.1.1 and later.
- Finished implementing auth for 4.0, 4.1.0, and 4.1.1.
- Changed namespace from `MySQL.Data.MySQLClient` back to `MySQL.Data.MySqlClient`.
- Fixed bug in `CharSetMapping` where it was trying to use text names as ints.
- Changed namespace to `MySQL.Data.MySQLClient`.
- Integrated auth changes from UC2004.

- Fixed bug where calling any of the GetXXX methods on a datareader before or after reading data would not throw the appropriate exception (thanks Luca Morelli).
- Added [TimeSpan](#) code in parameter.cs to properly serialize a timespan object to mysql time format (thanks Gianluca Colombo).
- Added [TimeStamp](#) to parameter serialization code. Prevented [DataAdatper](#) updates from working right (thanks Michael King).
- Fixed a misspelling in [MySQLHelper.cs](#) (thanks Patrick Kristiansen).

C.3.27. Changes in MySQL Connector/NET Version 0.76

- Driver now using charset number given in handshake to create encoding.
- Changed command editor to point to [MySQLClient.Design](#).
- Fixed bug in [Version.isAtLeast](#).
- Changed [DBConnectionString](#) to support changes done to [MySQLConnectionString](#).
- Removed [SqlCommandEditor](#) and [DataAdapterPreviewDialog](#).
- Using new long return values in many places.
- Integrated new [CompressedStream](#) class.
- Changed [ConnectionString](#) and added attributes to allow it to be used in [MySQLClient.Design](#).
- Changed [packet.cs](#) to support newer lengths in [ReadLenInteger](#).
- Changed other classes to use new properties and fields of [MySQLConnectionString](#).
- [ConnectionInternal](#) is now using PING to see whether the server is alive.
- Moved toolbox bitmaps into resource folder.
- Changed [field.cs](#) to allow values to come directly from row buffer.
- Changed to use the new driver.Send syntax.
- Using a new packet queueing system.
- Started work handling the "broken" compression packet handling.
- Fixed bug in [StreamCreator](#) where failure to connect to a host would continue to loop infinitely (thanks Kevin Casella).
- Improved connectstring handling.
- Moved designers into Pro product.
- Removed some old commented out code from [command.cs](#).
- Fixed a problem with compression.
- Fixed connection object where an exception throw prior to the connection opening would not leave the connection in the connecting state (thanks Chris Cline).
- Added GUID support.
- Fixed sequence out of order bug (thanks Mark Reay).

C.3.28. Changes in MySQL Connector/NET Version 0.75

- Enum values now supported as parameter values (thanks Philipp Sumi).
- Year datatype now supported.
- Fixed compression.
- Fixed bug where a parameter with a `TimeSpan` as the value would not serialize properly.
- Fixed bug where default constructor would not set default connection string values.
- Added some XML comments to some members.
- Work to fix/improve compression handling.
- Improved `ConnectionString` handling so that it better matches the standard set by `SqlClient`.
- A `MySqlException` is now thrown if a username is not included in the connection string.
- Localhost is now used as the default if not specified on the connection string.
- An exception is now thrown if an attempt is made to set the connection string while the connection is open.
- Small changes to `ConnectionString` docs.
- Removed `MultiHostStream` and `MySqlStream`. Replaced it with `Common/StreamCreator`.
- Added support for Use Pipe connection string value.
- Added Platform class for easier access to platform utility functions.
- Fixed small pooling bug where new connection was not getting created after `IsAlive` fails.
- Added `Platform.cs` and `StreamCreator.cs`.
- Fixed `Field.cs` to properly handle 4.1 style timestamps.
- Changed `Common.Version` to `Common.DBVersion` to avoid name conflict.
- Fixed `field.cs` so that text columns return the right field type.
- Added `MySqlError` class to provide some reference for error codes (thanks Geert Veenstra).

C.3.29. Changes in MySQL Connector/NET Version 0.74

- Added Unix socket support (thanks Mohammad DAMT).
- Only calling `Thread.Sleep` when no data is available.
- Improved escaping of quote characters in parameter data.
- Removed misleading comments from `parameter.cs`.
- Fixed pooling bug.
- Fixed `ConnectionString` editor dialog (thanks marco p (pomarc)).
- `UserId` now supported in connection strings (thanks Jeff Neeley).
- Attempting to create a parameter that is not input throws an exception (thanks Ryan Gregg).
- Added much documentation.
- Checked in new `MultiHostStream` capability. Big thanks to Dan Guisinger for this. he originally submitted the code and idea of supporting multiple machines on the connect string.

- Added a lot of documentation.
- Fixed speed issue with 0.73.
- Changed to `Thread.Sleep(0)` in `MySqlDataStream` to help optimize the case where it doesn't need to wait (thanks Todd German).
- Prepopulating the idlepools to `MinPoolSize`.
- Fixed `MySqlPool` deadlock condition as well as stupid bug where `CreateNewPooledConnection` was not ever adding new connections to the pool. Also fixed `MySqlStream.ReadBytes` and `ReadByte` to not use `TicksPerSecond` which does not appear to always be right. (thanks Matthew J. Peddlesden)
- Fix for precision and scale (thanks Matthew J. Peddlesden).
- Added `Thread.Sleep(1)` to stream reading methods to be more cpu friendly (thanks Sean McGinnis).
- Fixed problem where `ExecuteReader` would sometime return null (thanks Lloyd Dupont).
- Fixed major bug with null field handling (thanks Naucki).
- Enclosed queries for `max_allowed_packet` and `characterset` inside try catch (and set defaults).
- Fixed problem where socket was not getting closed properly (thanks Steve!).
- Fixed problem where `ExecuteNonQuery` was not always returning the right value.
- Fixed `InternalConnection` to not use `@@session.max_allowed_packet` but use `@@max_allowed_packet`. (Thanks Miguel)
- Added many new XML doc lines.
- Fixed sql parsing to not send empty queries (thanks Rory).
- Fixed problem where the reader was not unpeeking the packet on close.
- Fixed problem where user variables were not being handled (thanks Sami Vaaraniemi).
- Fixed loop checking in the `MySqlPool` (thanks Steve M. Brown)
- Fixed `ParameterCollection.Add` method to match `SqlClient` (thanks Joshua Mouch).
- Fixed `ConnectionString` parsing to handle no and yes for boolean and not lowercase values (thanks Naucki).
- Added `InternalConnection` class, changes to pooling.
- Implemented Persist Security Info.
- Added `security.cs` and `version.cs` to project
- Fixed `DateTime` handling in `Parameter.cs` (thanks Burkhard Perken-Golomb).
- Fixed parameter serialization where some types would throw a cast exception.
- Fixed `DataReader` to convert all returned values to prevent casting errors (thanks Keith Murray).
- Added code to `Command.ExecuteReader` to return null if the initial SQL command throws an exception (thanks Burkhard Perken-Golomb).
- Fixed `ExecuteScalar` bug introduced with restructure.
- Restructure to allow for `LOCAL DATA INFILE` and better sequencing of packets.
- Fixed several bugs related to restructure.
- Early work done to support more secure passwords in Mysql 4.1. Old passwords in 4.1 not supported yet.
- Parameters appearing after system parameters are now handled correctly (Adam M. (adammil)).

- Strings can now be assigned directly to blob fields (Adam M.).
- Fixed float parameters (thanks Pent).
- Improved Parameter constructor and `ParameterCollection.Add` methods to better match `SqlClient` (thanks Joshua Mouch).
- Corrected `Connection.CreateCommand` to return a `MySqlCommand` type.
- Fixed connection string designer dialog box problem (thanks Abraham Guxt).
- Fixed problem with sending commands not always reading the response packet (thanks Joshua Mouch).
- Fixed parameter serialization where some blobs types were not being handled (thanks Sean McGinnis).
- Removed spurious `MessageBox.show` from `DataReader` code (thanks Joshua Mouch).
- Fixed a nasty bug in the split sql code (thanks everyone!).

C.3.30. Changes in MySQL Connector/NET Version 0.71

- Fixed bug in `MySqlStream` where too much data could attempt to be read (thanks Peter Belbin)
- Implemented `HasRows` (thanks Nash Pherson).
- Fixed bug where tables with more than 252 columns cause an exception (thanks Joshua Kessler).
- Fixed bug where SQL statements ending in ; would cause a problem (thanks Shane Krueger).
- Fixed bug in driver where error messages were getting truncated by 1 character (thanks Shane Krueger).
- Made `MySqlException` serializable (thanks Mathias Hasselmann).

C.3.31. Changes in MySQL Connector/NET Version 0.70

- Updated some of the character code pages to be more accurate.
- Fixed problem where readers could be opened on connections that had readers open.
- Moved test to separate assembly `MySqlClientTests`.
- Fixed stupid problem in driver with sequence out of order (Thanks Peter Belbin).
- Added some pipe tests.
- Increased default max pool size to 50.
- Compiles with Mono 0-24.
- Fixed connection and data reader dispose problems.
- Added `String` datatype handling to parameter serialization.
- Fixed sequence problem in driver that occurred after thrown exception (thanks Burkhard Perens-Golomb).
- Added support for `CommandBehavior.SingleRow` to `DataReader`.
- Fixed command sql processing so quotes are better handled (thanks Theo Spears).
- Fixed parsing of double, single, and decimal values to account for non-English separators. You still have to use the right syntax if you using hard coded sql, but if you use parameters the code will convert floating point types to use '.' appropriately internal both in- to the server and out.

- Added [MySQLStream](#) class to simplify timeouts and driver coding.
- Fixed [DataReader](#) so that it is closed properly when the associated connection is closed. [thanks smishra]
- Made client more [SqlClient](#) compliant so that [DataReaders](#) have to be closed before the connection can be used to run another command.
- Improved [DBNull.Value](#) handling in the fields.
- Added several unit tests.
- Fixed [MySQLException](#) base class.
- Improved driver coding
- Fixed bug where [NextResult](#) was returning false on the last resultset.
- Added more tests for MySQL.
- Improved casting problems by equating unsigned 32bit values to [Int64](#) and unsigned 16bit values to [Int32](#), and so forth.
- Added new constructor for [MySQLParameter](#) for (name, type, size, srccol)
- Fixed bug in [MySQLDataReader](#) where it didn't check for null fieldlist before returning field count.
- Started adding [MySQLClient](#) unit tests (added [MySQLClient/Tests](#) folder and some test cases).
- Fixed some things in Connection String handling.
- Moved [INIT_DB](#) to [MySQLPool](#). I may move it again, this is in preparation of the conference.
- Fixed bug inside [CommandBuilder](#) that prevented inserts from happening properly.
- Reworked some of the internals so that all three execute methods of [Command](#) worked properly.
- Fixed many small bugs found during benchmarking.
- The first cut of [CoonnectionPooling](#) is working. "min pool size" and "max pool size" are respected.
- Work to enable multiple resultsets to be returned.
- Character sets are handled much more intelligently now. The driver queries MySQL at startup for the default character set. That character set is then used for conversions if that code page can be loaded. If not, then the default code page for the current OS is used.
- Added code to save the inferred type in the name,value constructor of [Parameter](#).
- Also, inferred type if value of null parameter is changed using [Value](#) property.
- Converted all files to use proper Camel case. MySQL is now [MySql](#) in all files. [PgSQL](#) is now [PgSql](#).
- Added attribute to [PgSql](#) code to prevent designer from trying to show.
- Added [MySQLDbType](#) property to [Parameter](#) object and added proper conversion code to convert from [DbType](#) to [MySQLDbType](#)).
- Removed unused [ObjectToString](#) method from [MySQLParameter.cs](#).
- Fixed [Add\(. . \)](#) method in [ParameterCollection](#) so that it doesn't use [Add\(name, value\)](#) instead.
- Fixed [IndexOf](#) and [Contains](#) in [ParameterCollection](#) to be aware that parameter names are now stored without @.
- Fixed [Command.ConvertSQLToBytes](#) so it only allows characters that can be in MySQL variable names.
- Fixed [DataReader](#) and [Field](#) so that blob fields read their data from [Field.cs](#) and [GetBytes](#) works right.
- Added simple query builder editor to [CommandText](#) property of [MySQLCommand](#).

- Fixed `CommandBuilder` and `Parameter` serialization to account for Parameters not storing @ in their names.
- Removed `MySQLFieldType` enum from `Field.cs`. Now using `MySQLDbType` enum.
- Added `Designer` attribute to several classes to prevent designer view when using VS.Net.
- Fixed Initial catalog typo in `ConnectionString` designer.
- Removed 3 parameter constructor for `MySQLParameter` that conflicted with (name, type, value).
- Changed `MySQLParameter` so `paramName` is now stored without leading @ (this fixed null inserts when using designer).
- Changed `TypeConverter` for `MySQLParameter` to use the constructor with all properties.

C.3.32. Changes in MySQL Connector/NET Version 0.68

- Fixed sequence issue in driver.
- Added `DbParametersEditor` to make parameter editing more like `SqlClient`.
- Fixed `Command` class so that parameters can be edited using the designer
- Update connection string designer to support `Use Compression` flag.
- Fixed string encoding so that European characters like ä will work correctly.
- Creating base classes to aid in building new data providers.
- Added support for UID key in connection string.
- Field, parameter, command now using `DBNull.Value` instead of null.
- `CommandBuilder` using `DBNull.Value`.
- `CommandBuilder` now builds insert command correctly when an `auto_insert` field is not present.
- Field now uses `typeof` keyword to return `System.Types` (performance).

C.3.33. Changes in MySQL Connector/NET Version 0.65

- `MySQLCommandBuilder` now implemented.
- Transaction support now implemented (not all table types support this).
- `GetSchemaTable` fixed to not use `xsd` (for Mono).
- Driver is now Mono-compatible.
- TIME data type now supported.
- More work to improve Timestamp data type handling.
- Changed signatures of all classes to match corresponding `SqlClient` classes.

C.3.34. Changes in MySQL Connector/NET Version 0.60

- Protocol compression using SharpZipLib (www.icsharpcode.net).
- Named pipes on Windows now working properly.

- Work done to improve [Timestamp](#) data type handling.
- Implemented [IEnumerable](#) on [DataReader](#) so [DataGrid](#) would work.

C.3.35. Changes in MySQL Connector/NET Version 0.50

- Speed increased dramatically by removing bugging network sync code.
- Driver no longer buffers rows of data (more ADO.Net compliant).
- Conversion bugs related to [TIMESTAMP](#) and [DATETIME](#) fields fixed.

C.4. MySQL Visual Studio Plugin Change History

Note

As of Connector/NET 5.1.2 (14 June 2007), the Visual Studio Plugin is part of the main Connector/NET package. For the change history for the Visual Studio Plugin, see [Section C.3, "Connector/NET Change History"](#).

C.4.1. Changes in MySQL Visual Studio Plugin 1.1.3 (Not yet released)

Bugs fixed:

- Running queries based on a stored procedure would cause the dataset designer to terminate. (Bugs #26364)
- DataSet wizard would show all tables instead of only the tables available within the selected database. (Bugs #26348)

C.4.2. Changes in MySQL Visual Studio Plugin 1.0.2 (Not yet released)

Bugs fixed:

- Creating a connection through the Server Explorer when using the Visual Studio Plugin would fail. The installer for the Visual Studio Plugin has been updated to ensure that Connector/NET 5.0.2 must be installed. ([Bug#23071](#))
- The Add Connection dialog of the Server Explorer would freeze when accessing databases with capitalized characters in their name. ([Bug#24875](#))

C.4.3. Changes in MySQL Visual Studio Plugin 1.0.1 (4 October 2006)

This is a bug fix release to resolve an incompatibility issue with Connector/NET 5.0.1.

It is critical that this release only be used with Connector/NET 5.0.1. After installing Connector/NET 5.0.1, you will need to make a small change in your machine.config file. This file should be located at `%win%\Microsoft .Net\Framework\v2.0.50727\CONFIG\machine.config` (`%win%` should be the location of your Windows folder). Near the bottom of the file you will see a line like this:

```
<add name="MySQL Data Provider" invariant="MySQL.Data.MySqlClient"
description=".Net Framework Data Provider for MySQL"
type="MySQL.Data.MySqlClient.MySqlClientFactory, MySQL.Data"/>
```

It needs to be changed to be like this:

```
<add name="MySQL Data Provider" invariant="MySQL.Data.MySqlClient"
description=".Net Framework Data Provider for MySQL"
type="MySQL.Data.MySqlClient.MySqlClientFactory, MySQL.Data,
Version=5.0.1.0, Culture=neutral, PublicKeyToken=c5687fc88969c44d"/>
```

C.4.4. Changes in MySQL Visual Studio Plugin 1.0.0 (4 October 2006)

This is the first beta release.

Features in this release:

- DDEX (Data Designer Extensibility) compatibility.
- Ability to work with MySQL objects (tables, views, stored procedures, etc) from within Server Explorer.

C.5. MySQL Connector/J Change History

C.5.1. Changes in MySQL Connector/J 5.1.x

C.5.1.1. Changes in MySQL Connector/J 5.1.3 (10 September 2007)

Functionality added or changed:

- Connector/J now connects using an initial character set of **utf-8** solely for the purpose of authentication to allow user names or database names in any character set to be used in the JDBC connection URL. ([Bug#29853](#))
- Added two configuration parameters:
 - **blobsAreStrings** Should the driver always treat BLOBs as Strings. Added specifically to work around dubious metadata returned by the server for **GROUP BY** clauses. Defaults to false.
 - **functionsNeverReturnBlobs** Should the driver always treat data from functions returning **BLOBs** as Strings. Added specifically to work around dubious metadata returned by the server for **GROUP BY** clauses. Defaults to false.
- Setting **rewriteBatchedStatements** to **true** now causes CallableStatements with batched arguments to be re-written in the form "CALL (...); CALL (...); ..." to send the batch in as few client-server round trips as possible.
- The driver now picks appropriate internal row representation (whole row in one buffer, or individual byte[]s for each column value) depending on heuristics, including whether or not the row has **BLOB** or **TEXT** types and the overall row-size. The threshold for row size that will cause the driver to use a buffer rather than individual byte[]s is configured by the configuration property **largeRowSizeThreshold**, which has a default value of 2KB.
- The data (and how it's stored) for **ResultSet** rows are now behind an interface which allows us (in some cases) to allocate less memory per row, in that for "streaming" result sets, we re-use the packet used to read rows, since only one row at a time is ever active.
- Added experimental support for statement "interceptors" via the **com.mysql.jdbc.StatementInterceptor** interface, examples are in [com/mysql/jdbc/interceptors](#). Implement this interface to be placed "in between" query execution, so that it can be influenced (currently experimental).
- The driver will automatically adjust the server session variable **net_write_timeout** when it determines its been asked for a "streaming" result, and resets it to the previous value when the result set has been consumed. (The configuration property is named **netTimeoutForStreamingResults**, with a unit of seconds, the value '0' means the driver will not try and adjust this value).
- Support for JDBC-4.0 **NCHAR**, **NVARCHAR** and **NCLOB** types.
- JDBC-4.0 support for setting per-connection client information (which can be viewed in the comments section of a query via **SHOW PROCESSLIST** on a MySQL server, or can be extended to support custom persistence of the information via a public interface).
- Support for JDBC-4.0 XML processing via JAXP interfaces to DOM, SAX and StAX.
- JDBC-4.0 standardized unwrapping to interfaces that include vendor extensions.
- JDBC-4.0 ease-of-development features including auto-registration with the **DriverManager** via the service provider mechanism, standardized Connection validity checks and categorized **SQLExceptions** based on recoverability/retry-ability and class of the underlying error.

- `Statement.setQueryTimeout()`s now affect the entire batch for batched statements, rather than the individual statements that make up the batch.
- Errors encountered during `Statement/PreparedStatement/CallableStatement.executeBatch()` when `rewriteBatchStatements` has been set to `true` now return `BatchUpdateExceptions` according to the setting of `continueBatchOnError`.

If `continueBatchOnError` is set to `true`, the update counts for the "chunk" that were sent as one unit will all be set to `EXECUTE_FAILED`, but the driver will attempt to process the remainder of the batch. You can determine which "chunk" failed by looking at the update counts returned in the `BatchUpdateException`.

If `continueBatchOnError` is set to `false`, the update counts returned will contain all updates up-to and including the failed "chunk", with all counts for the failed "chunk" set to `EXECUTE_FAILED`.

Since MySQL doesn't return multiple error codes for multiple-statements, or for multi-value `INSERT/REPLACE`, it is the application's responsibility to handle determining which item(s) in the "chunk" actually failed.

- New methods on `com.mysql.jdbc.Statement`: `setLocalInfileInputStream()` and `getLocalInfileInputStream()`:
 - `setLocalInfileInputStream()` sets an `InputStream` instance that will be used to send data to the MySQL server for a `LOAD DATA LOCAL INFILE` statement rather than a `FileInputStream` or `URLInputStream` that represents the path given as an argument to the statement.

This stream will be read to completion upon execution of a `LOAD DATA LOCAL INFILE` statement, and will automatically be closed by the driver, so it needs to be reset before each call to `execute*()` that would cause the MySQL server to request data to fulfill the request for `LOAD DATA LOCAL INFILE`.

If this value is set to `NULL`, the driver will revert to using a `FileInputStream` or `URLInputStream` as required.

- `getLocalInfileInputStream()` returns the `InputStream` instance that will be used to send data in response to a `LOAD DATA LOCAL INFILE` statement.

This method returns `NULL` if no such stream has been set via `setLocalInfileInputStream()`.

- Setting `useBlobToStoreUTF8OutsideBMP` to `true` tells the driver to treat `[MEDIUM/LONG]BLOB` columns as `[LONG]VARCHAR` columns holding text encoded in UTF-8 that has characters outside the BMP (4-byte encodings), which MySQL server can't handle natively.

Set `utf8OutsideBmpExcludedColumnNamePattern` to a regex so that column names matching the given regex will still be treated as `BLOBs`. The regex must follow the patterns used for the `java.util.regex` package. The default is to exclude no columns, and include all columns.

Set `utf8OutsideBmpIncludedColumnNamePattern` to specify exclusion rules to `utf8OutsideBmpExcludedColumnNamePattern`. The regex must follow the patterns used for the `java.util.regex` package.

Bugs fixed:

- `setObject(int, Object, int, int)` delegate in `PreparedStatementWrapper` delegates to wrong method. ([Bug#30892](#))
- NPE with null column values when `padCharsWithSpace` is set to true. ([Bug#30851](#))
- Collation on `VARBINARY` column types would be misidentified. A fix has been added, but this fix only works for MySQL server versions 5.0.25 and newer, since earlier versions didn't consistently return correct metadata for functions, and thus results from sub-queries and functions were indistinguishable from each other, leading to type-related bugs. ([Bug#30664](#))
- An `ArithmeticException` or `NullPointerException` would be raised when the batch had zero members and `rewriteBatchedStatements=true` when `addBatch()` was never called, or `executeBatch()` was called immediately after `clearBatch()`. ([Bug#30550](#))
- Closing a load-balanced connection would cause a `ClassCastException`. ([Bug#29852](#))
- Connection checker for JBoss didn't use same method parameters via reflection, causing connections to always seem "bad". ([Bug#29106](#))

- `DatabaseMetaData.getTypeInfo()` for the types `DECIMAL` and `NUMERIC` will return a precision of 254 for server versions older than 5.0.3, 64 for versions 5.0.3-5.0.5 and 65 for versions newer than 5.0.5. ([Bug#28972](#))
- `CallableStatement.executeBatch()` doesn't work when connection property `noAccessToProcedureBodies` has been set to `true`.

The fix involves changing the behavior of `noAccessToProcedureBodies`, in that the driver will now report all parameters as "IN" parameters but allow callers to call `registerOutParameter()` on them without throwing an exception. ([Bug#28689](#))
- `DatabaseMetaData.getColumns()` doesn't contain `SCOPE_*` or `IS_AUTOINCREMENT` columns. ([Bug#27915](#))
- Schema objects with identifiers other than the connection character aren't retrieved correctly in `ResultSetMetadata`. ([Bug#27867](#))
- `Connection.getServerCharacterEncoding()` doesn't work for servers with version ≥ 4.1 . ([Bug#27182](#))
- The automated SVN revisions in `DBMD.getDriverVersion()`. The SVN revision of the directory is now inserted into the version information during the build. ([Bug#21116](#))
- Specifying a "validation query" in your connection pool that starts with `"/ * ping */"` _exactly_ will cause the driver to instead send a ping to the server and return a fake result set (much lighter weight), and when using a `ReplicationConnection` or a `LoadBalancedConnection`, will send the ping across all active connections.

C.5.1.2. Changes in MySQL Connector/J 5.1.2 (29 June 2007)

This is a new Beta development release, fixing recently discovered bugs.

Functionality added or changed:

- Setting the configuration property `rewriteBatchedStatements` to `true` will now cause the driver to rewrite batched prepared statements with more than 3 parameter sets in a batch into multi-statements (separated by `;`) if they are not plain (*i.e.* without `SELECT` or `ON DUPLICATE KEY UPDATE` clauses) `INSERT` or `REPLACE` statements.

C.5.1.3. Changes in MySQL Connector/J 5.1.1 (22 June 2007)

This is a new Alpha development release, adding new features and fixing recently discovered bugs.

Functionality added or changed:

- **Incompatible change:** Pulled vendor-extension methods of `Connection` implementation out into an interface to support `java.sql.Wrapper` functionality from `ConnectionPoolDataSource`. The vendor extensions are javadoc'd in the `com.mysql.jdbc.Connection` interface.

For those looking further into the driver implementation, it is not an API that is used for pluggability of implementations inside our driver (which is why there are still references to `ConnectionImpl` throughout the code).

We've also added server and client `prepareStatement()` methods that cover all of the variants in the JDBC API.

`Connection.serverPrepare(String)` has been re-named to `Connection.serverPrepareStatement()` for consistency with `Connection.clientPrepareStatement()`.

- Row navigation now causes any streams/readers open on the result set to be closed, as in some cases we're reading directly from a shared network packet and it will be overwritten by the "next" row.
- Made it possible to retrieve prepared statement parameter bindings (to be used in `StatementInterceptors`, primarily).
- Externalized the descriptions of connection properties.
- The data (and how it's stored) for `ResultSet` rows are now behind an interface which allows us (in some cases) to allocate less memory per row, in that for "streaming" result sets, we re-use the packet used to read rows, since only one row at a time is ever active.

- Similar to [Connection](#), we pulled out vendor extensions to [Statement](#) into an interface named [com.mysql.Statement](#), and moved the [Statement](#) class into [com.mysql.StatementImpl](#). The two methods (javadoc'd in [com.mysql.Statement](#) are [enableStreamingResults\(\)](#), which already existed, and [disableStreamingResults\(\)](#) which sets the statement instance back to the fetch size and result set type it had before [enableStreamingResults\(\)](#) was called.
- Driver now picks appropriate internal row representation (whole row in one buffer, or individual byte[]s for each column value) depending on heuristics, including whether or not the row has [BLOB](#) or [TEXT](#) types and the overall row-size. The threshold for row size that will cause the driver to use a buffer rather than individual byte[]s is configured by the configuration property [largeRowSizeThreshold](#), which has a default value of 2KB.
- Added experimental support for statement "interceptors" via the [com.mysql.jdbc.StatementInterceptor](#) interface, examples are in [com/mysql/jdbc/interceptors](#).

Implement this interface to be placed "in between" query execution, so that you can influence it. (currently experimental).

[StatementInterceptors](#) are "chainable" when configured by the user, the results returned by the "current" interceptor will be passed on to the next on in the chain, from left-to-right order, as specified by the user in the JDBC configuration property [statementInterceptors](#).

- See the sources (fully javadoc'd) for [com.mysql.jdbc.StatementInterceptor](#) for more details until we iron out the API and get it documented in the manual.
- Setting [rewriteBatchedStatements](#) to [true](#) now causes [CallableStatements](#) with batched arguments to be rewritten in the form [CALL \(...\); CALL \(...\); ...](#) to send the batch in as few client-server round trips as possible.

C.5.1.4. Changes in MySQL Connector/J 5.1.0 (11 April 2007)

This is the first public alpha release of the current Connector/J 5.1 development branch, providing an insight to upcoming features. Although some of these are still under development, this release includes the following new features and changes (in comparison to the current Connector/J 5.0 production release):

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is [false](#) (that is, Connector/J does not use server-side prepared statements).

Note

The disabling of server-side prepared statements does not affect the operation of the connector. However, if you use the [useTimezone=true](#) connection option and use client-side prepared statements (instead of server-side prepared statements) you should also set [useSSPSCompatibleTimezoneShift=true](#).

Functionality added or changed:

- Refactored [CommunicationsException](#) into a JDBC-3.0 version, and a JDBC-4.0 version (which extends [SQLRecoverableException](#), now that it exists).

Note

This change means that if you were catching [com.mysql.jdbc.CommunicationsException](#) in your applications instead of looking at the [SQLState](#) class of [08](#), and are moving to Java 6 (or newer), you need to change your imports to that exception to be [com.mysql.jdbc.exceptions.jdbc4.CommunicationsException](#), as the old class will not be instantiated for communications link-related errors under Java 6.

- Added support for JDBC-4.0 categorized [SQLExceptions](#).
- Added support for JDBC-4.0's [NCLOB](#), and [NCHAR/NVARCHAR](#) types.

- `com.mysql.jdbc.java6.javac` full path to your Java-6 `javac` executable
- Added support for JDBC-4.0's SQLXML interfaces.
- Re-worked Ant buildfile to build JDBC-4.0 classes separately, as well as support building under Eclipse (since Eclipse can't mix/match JDKs).

To build, you must set `JAVA_HOME` to J2SDK-1.4.2 or Java-5, and set the following properties on your Ant command line:

 - `com.mysql.jdbc.java6.javac` full path to your Java-6 `javac` executable
 - `com.mysql.jdbc.java6.rtjar` full path to your Java-6 `rt.jar` file
- New feature driver will automatically adjust session variable `net_write_timeout` when it determines it has been asked for a "streaming" result, and resets it to the previous value when the result set has been consumed. (configuration property is named `netTimeoutForStreamingResults` value and has a unit of seconds, the value `0` means the driver will not try and adjust this value).
- Added support for JDBC-4.0's client information. The backend storage of information provided via `Connection.setClientInfo()` and retrieved by `Connection.getClientInfo()` is pluggable by any class that implements the `com.mysql.jdbc.JDBC4ClientInfoProvider` interface and has a no-args constructor.

The implementation used by the driver is configured using the `clientInfoProvider` configuration property (with a default of value of `com.mysql.jdbc.JDBC4CommentClientInfoProvider`, an implementation which lists the client information as a comment prepended to every query sent to the server).

This functionality is only available when using Java-6 or newer.
- `com.mysql.jdbc.java6.rtjar` full path to your Java-6 `rt.jar` file
- Added support for JDBC-4.0's `Wrapper` interface.

C.5.2. Changes in MySQL Connector/J 5.0.x

C.5.2.1. Changes in MySQL Connector/J 5.0.8 (Not yet released)

Functionality added or changed:

- `blobsAreStrings` Should the driver always treat BLOBs as Strings. Added specifically to work around dubious metadata returned by the server for `GROUP BY` clauses. Defaults to false.
- Added two configuration parameters:
 - `blobsAreStrings` Should the driver always treat BLOBs as Strings. Added specifically to work around dubious metadata returned by the server for `GROUP BY` clauses. Defaults to false.
 - `functionsNeverReturnBlobs` Should the driver always treat data from functions returning `BLOBs` as Strings. Added specifically to work around dubious metadata returned by the server for `GROUP BY` clauses. Defaults to false.
- `functionsNeverReturnBlobs` Should the driver always treat data from functions returning `BLOBs` as Strings. Added specifically to work around dubious metadata returned by the server for `GROUP BY` clauses. Defaults to false.

Bugs fixed:

- `setObject(int, Object, int, int)` delegate in `PreparedStatementWrapper` delegates to wrong method. ([Bug#30892](#))
- NPE with null column values when `padCharsWithSpace` is set to true. ([Bug#30851](#))
- Collation on `VARBINARY` column types would be misidentified. A fix has been added, but this fix only works for MySQL server versions 5.0.25 and newer, since earlier versions didn't consistently return correct metadata for functions, and thus results from sub-queries and functions were indistinguishable from each other, leading to type-related bugs. ([Bug#30664](#))

- An `ArithmeticException` or `NullPointerException` would be raised when the batch had zero members and `rewriteBatchedStatements=true` when `addBatch()` was never called, or `executeBatch()` was called immediately after `clearBatch()`. (Bug#30550)
- Closing a load-balanced connection would cause a `ClassCastException`. (Bug#29852)
- Connection checker for JBoss didn't use same method parameters via reflection, causing connections to always seem "bad". (Bug#29106)
- `DatabaseMetaData.getTypeInfo()` for the types `DECIMAL` and `NUMERIC` will return a precision of 254 for server versions older than 5.0.3, 64 for versions 5.0.3-5.0.5 and 65 for versions newer than 5.0.5. (Bug#28972)
- `CallableStatement.executeBatch()` doesn't work when connection property `noAccessToProcedureBodies` has been set to `true`.

The fix involves changing the behavior of `noAccessToProcedureBodies`, in that the driver will now report all parameters as "IN" parameters but allow callers to call `registerOutParameter()` on them without throwing an exception. (Bug#28689)

- `DatabaseMetaData.getColumns()` doesn't contain `SCOPE_*` or `IS_AUTOINCREMENT` columns. (Bug#27915)
- Schema objects with identifiers other than the connection character aren't retrieved correctly in `ResultSetMetadata`. (Bug#27867)
- `Connection.getServerCharacterEncoding()` doesn't work for servers with version ≥ 4.1 . (Bug#27182)
- The automated SVN revisions in `DBMD.getDriverVersion()`. The SVN revision of the directory is now inserted into the version information during the build. (Bug#21116)
- Specifying a "validation query" in your connection pool that starts with `"/ * ping * /"` _exactly_ will cause the driver to instead send a ping to the server and return a fake result set (much lighter weight), and when using a `ReplicationConnection` or a `LoadBalancedConnection`, will send the ping across all active connections.

C.5.2.2. Changes in MySQL Connector/J 5.0.7 (20 July 2007)

Functionality added or changed:

- Setting the configuration parameter `useCursorFetch` to `true` for MySQL-5.0+ enables the use of cursors that allow Connector/J to save memory by fetching result set rows in chunks (where the chunk size is set by calling `setFetchSize()` on a `Statement` or `ResultSet`) by using fully-materialized cursors on the server.
- The driver will now automatically set `useServerPrepStmts` to `true` when `useCursorFetch` has been set to `true`, since the feature requires server-side prepared statements in order to function.
- Added configuration property `useNanosForElapsedTime` - for profiling/debugging functionality that measures elapsed time, should the driver try to use nanoseconds resolution if available (requires JDK ≥ 1.5)?

Note

If `useNanosForElapsedTime` is set to `true`, and this property is set to "0" (or left default), then elapsed times will still be measured in nanoseconds (if possible), but the slow query threshold will be converted from milliseconds to nanoseconds, and thus have an upper bound of approximately 2000 milliseconds (as that threshold is represented as an integer, not a long).

- Added configuration property `slowQueryThresholdNanos` - if `useNanosForElapsedTime` is set to `true`, and this property is set to a non-zero value the driver will use this threshold (in nanosecond units) to determine if a query was slow, instead of using millisecond units.
- Added configuration properties to allow tuning of TCP/IP socket parameters:
 - `tcpNoDelay` - Should the driver set `SO_TCP_NODELAY` (disabling the Nagle Algorithm, default `true`)?
 - `tcpKeepAlive` - Should the driver set `SO_KEEPALIVE` (default `true`)?
 - `tcpRcvBuf` - Should the driver set `SO_RCV_BUF` to the given value? The default value of '0', means use the platform default

value for this property.

- `tcpSndBuf` - Should the driver set `SO_SND_BUF` to the given value? The default value of '0', means use the platform default value for this property.
- `tcpTrafficClass` - Should the driver set traffic class or type-of-service fields? See the documentation for `java.net.Socket.setTrafficClass()` for more information.
- Give more information in EOFExceptions thrown out of `MysqlIO` (how many bytes the driver expected to read, how many it actually read, say that communications with the server were unexpectedly lost).
- Setting `useDynamicCharsetInfo` to `false` now causes driver to use static lookups for collations as well (makes `ResultSetMetadata.isCaseSensitive()` much more efficient, which leads to performance increase for ColdFusion, which calls this method for every column on every table it sees, it appears).
- Driver detects when it is running in a ColdFusion MX server (tested with version 7), and uses the configuration bundle `coldFusion`, which sets `useDynamicCharsetInfo` to `false` (see previous entry), and sets `useLocalSessionState` and `autoReconnect` to `true`.
- Added new debugging functionality - Setting configuration property `includeInnodbStatusInDeadlockExceptions` to `true` will cause the driver to append the output of `SHOW ENGINE INNODB STATUS` to deadlock-related exceptions, which will enumerate the current locks held inside InnoDB.

Bugs fixed:

- `PreparedStatement.getMetaData()` for statements containing leading one-line comments is not returned correctly.
As part of this fix, we also overhauled detection of DML for `executeQuery()` and `SELECTs` for `executeUpdate()` in plain and prepared statements to be aware of the same types of comments. ([Bug#28469](#))
- Parser in client-side prepared statements eats character following `'` if it's not a multi-line comment. ([Bug#28851](#))
- Parser in client-side prepared statements runs to end of statement, rather than end-of-line for `'#` comments. Also added support for `'-` single-line comments. ([Bug#28956](#))
- Don't send any file data in response to `LOAD DATA LOCAL INFILE` if the feature is disabled at the client side. This is to prevent a malicious server or man-in-the-middle from asking the client for data that the client is not expecting. Thanks to Jan Kneschke for discovering the exploit and Andrey "Poohie" Hristov, Konstantin Osipov and Sergei Golubchik for discussions about implications and possible fixes. ([Bug#29605](#))

C.5.2.3. Changes in MySQL Connector/J 5.0.6 (15 May 2007)

Functionality added or changed:

- Fixed issue where a failed-over connection would let an application call `setReadOnly(false)`, when that call should be ignored until the connection is reconnected to a writable master unless `failoverReadOnly` had been set to `false`.
- Driver will now use `INSERT INTO ... VALUES (DEFAULT)` form of statement for updatable result sets for `ResultSet.insertRow()`, rather than pre-populating the insert row with values from `DatabaseMetaData.getColumns()` (which results in a `SHOW FULL COLUMNS` on the server for every result set). If an application requires access to the default values before `insertRow()` has been called, the JDBC URL should be configured with `populateInsertRowWithDefaultValues` set to `true`.

This fix specifically targets performance issues with ColdFusion and the fact that it seems to ask for updatable result sets no matter what the application does with them.

- `com.mysql.jdbc.[NonRegistering]Driver` now understands URLs of the format `jdbc:mysql:replication://` and `jdbc:mysql:loadbalance://` which will create a `ReplicationConnection` (exactly like when using `[NonRegistering]ReplicationDriver`) and an experimental load-balanced connection designed for use with SQL nodes in a MySQL Cluster/NDB environment, respectively.

In an effort to simplify things, we're working on deprecating multiple drivers, and instead specifying different core behavior based upon JDBC URL prefixes, so watch for `[NonRegistering]ReplicationDriver` to eventually disappear, to be replaced with `com.mysql.jdbc[NonRegistering]Driver` with the new URL prefix.

- Added an experimental load-balanced connection designed for use with SQL nodes in a MySQL Cluster/NDB environment (This is not for master-slave replication. For that, we suggest you look at `ReplicationConnection` or `lbpool`).

If the JDBC URL starts with `jdbc:mysql:loadbalance://host-1,host-2,...host-n`, the driver will create an implementation of `java.sql.Connection` that load balances requests across a series of MySQL JDBC connections to the given hosts, where the balancing takes place after transaction commit.

Therefore, for this to work (at all), you must use transactions, even if only reading data.

Physical connections to the given hosts will not be created until needed.

The driver will invalidate connections that it detects have had communication errors when processing a request. A new connection to the problematic host will be attempted the next time it is selected by the load balancing algorithm.

There are two choices for load balancing algorithms, which may be specified by the `loadBalanceStrategy` JDBC URL configuration property:

- `random` — the driver will pick a random host for each request. This tends to work better than round-robin, as the randomness will somewhat account for spreading loads where requests vary in response time, while round-robin can sometimes lead to overloaded nodes if there are variations in response times across the workload.
- `bestResponseTime` — the driver will route the request to the host that had the best response time for the previous transaction.
- When `useLocalSessionState` is set to `true` and connected to a MySQL-5.0 or later server, the JDBC driver will now determine whether an actual `commit` or `rollback` statement needs to be sent to the database when `Connection.commit()` or `Connection.rollback()` is called.

This is especially helpful for high-load situations with connection pools that always call `Connection.rollback()` on connection check-in/check-out because it avoids a round-trip to the server.

- Give better error message when "streaming" result sets, and the connection gets clobbered because of exceeding `net_write_timeout` on the server.
- New configuration property, `enableQueryTimeouts` (default `true`).

When enabled, query timeouts set via `Statement.setQueryTimeout()` use a shared `java.util.Timer` instance for scheduling. Even if the timeout doesn't expire before the query is processed, there will be memory used by the `TimerTask` for the given timeout which won't be reclaimed until the time the timeout would have expired if it hadn't been cancelled by the driver. High-load environments might want to consider disabling this functionality. (this configuration property is part of the `maxPerformance` configuration bundle).

- Added configuration property `padCharsWithSpace` (defaults to `false`). If set to `true`, and a result set column has the `CHAR` type and the value does not fill the amount of characters specified in the DDL for the column, the driver will pad the remaining characters with space (for ANSI compliance).
- Added configuration property `useDynamicCharsetInfo`. If set to `false` (the default), the driver will use a per-connection cache of character set information queried from the server when necessary, or when set to `true`, use a built-in static mapping that is more efficient, but isn't aware of custom character sets or character sets implemented after the release of the JDBC driver.

Note

This only affects the `padCharsWithSpace` configuration property and the `ResultSetMetaData.getColumnDisplayWidth()` method.

- More intelligent initial packet sizes for the "shared" packets are used (512 bytes, rather than 16K), and initial packets used during handshake are now sized appropriately as to not require reallocation.

Bugs fixed:

- More useful error messages are generated when the driver thinks a result set is not updatable. (Thanks to Ashley Martens for the patch). ([Bug#28085](#))
- Using `ResultSet.get*()` with a column index less than 1 returns a misleading error message. ([Bug#27317](#))
- Client options not sent correctly when using SSL, leading to stored procedures not being able to return results. Thanks to Don Cohen for the bug report, testcase and patch. ([Bug#25545](#))
- `PreparedStatement` is not closed in `BlobFromLocator.getBytes()`. ([Bug#26592](#))
- Whitespace surrounding storage/size specifiers in stored procedure parameters declaration causes `NumberFormatException` to be thrown when calling stored procedure on JDK-1.5 or newer, as the Number classes in JDK-1.5+ are whitespace intolerant. ([Bug#25624](#))
- When the configuration property `useCursorFetch` was set to `true`, sometimes server would return new, more exact metadata during the execution of the server-side prepared statement that enables this functionality, which the driver ignored (using the original metadata returned during `prepare()`), causing corrupt reading of data due to type mismatch when the actual rows were returned. ([Bug#26173](#))
- Comments in DDL of stored procedures/functions confuse procedure parser, and thus metadata about them can not be created, leading to inability to retrieve said metadata, or execute procedures that have certain comments in them. ([Bug#26959](#))
- Fast date/time parsing doesn't take into account `00:00:00` as a legal value. ([Bug#26789](#))
- `ResultSet.get*()` with a column index < 1 returns misleading error message. ([Bug#27317](#))
- `Statement.setMaxRows()` is not effective on result sets materialized from cursors. ([Bug#25517](#))
- `CALL /* ... */ some_proc()` doesn't work. As a side effect of this fix, you can now use `/* */` and `#` comments when preparing statements using client-side prepared statement emulation. ([Bug#27400](#))

If the comments happen to contain parameter markers (?), they will be treated as belonging to the comment (that is, not recognized) rather than being a parameter of the statement.

Note

The statement when sent to the server will contain the comments as-is, they're not stripped during the process of preparing the `PreparedStatement` or `CallableStatement`.

- `BIT(> 1)` is returned as `java.lang.String` from `ResultSet.getObject()` rather than `byte[]`. ([Bug#25328](#))
- `CallableStatements` with `OUT/INOUT` parameters that are "binary" (`BLOB`, `BIT`, `(VAR)BINARY`, `JAVA_OBJECT`) have extra 7 bytes. ([Bug#25715](#))
- `Connection.getTransactionIsolation()` uses "`SHOW VARIABLES LIKE`" which is very inefficient on MySQL-5.0+ servers. ([Bug#27655](#))
- Fixed issue where calling `getGeneratedKeys()` on a prepared statement after calling `execute()` didn't always return the generated keys (`executeUpdate()` worked fine however).

C.5.2.4. Changes in MySQL Connector/J 5.0.5 (02 March 2007)

Functionality added or changed:

- **Important change:** Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is `false` (that is, Connector/J does not use server-side prepared statements).

- Improved speed of `datetime` parsing for ResultSets that come from plain or non-server-side prepared statements. You can enable old implementation with `useFastDateParsing=false` as a configuration parameter.
- Added configuration property `localSocketAddress`, which is the hostname or IP address given to explicitly configure the interface that the driver will bind the client side of the TCP/IP connection to when connecting.
- The `rewriteBatchedStatements` feature can now be used with server-side prepared statements.
- We've added a new configuration option `treatUtilDateAsTimestamp`, which is `false` by default, as (1) We already had specific behavior to treat `java.util.Date` as a `java.sql.Timestamp` because it's useful to many folks, and (2) that behavior will very likely be required for drivers JDBC-post-4.0.
- Fixed logging of XA commands sent to server, it's now configurable via `logXaCommands` property (defaults to `false`).
- Usage Advisor now detects empty results sets and does not report on columns not referenced in those empty sets.
- Usage Advisor will now issue warnings for result sets with large numbers of rows. You can configure the trigger value by using the `resultSetSizeThreshold` parameter, which has a default value of 100.

Bugs fixed:

- Fixed an issue where `XADataSources` couldn't be bound into JNDI, as the `DataSourceFactory` didn't know how to create instances of them.
- Calling `Statement.cancel()` could result in a Null Pointer Exception (NPE). (Bug#24721)
- Calendars and timezones are now lazily instantiated when required. (Bug#24351)
- Client-side prepared statement parser gets confused by in-line comments `/* ... */` and therefore cannot rewrite batch statements or reliably detect the type of statements when they are used. (Bug#25025)
- When using a JDBC connection URL that is malformed, the `NonRegisteringDriver.getPropertyInfo` method will throw a Null Pointer Exception (NPE). (Bug#22628)
- Using `DatabaseMetaData.getSQLKeywords()` does not return a all of the of the reserved keywords for the current MySQL version. Current implementation returns the list of reserved words for MySQL 5.1, and does not distinguish between versions. (Bug#24794)
- Specifying `US-ASCII` as the character set in a connection to a MySQL 4.1 or newer server does not map correctly. (Bug#24840)
- Storing a `java.util.Date` object in a `BLOB` column would not be serialized correctly during `setObject`. (Bug#25787)
- A query execution which timed out did not always throw a `MySQLTimeoutException`. (Bug#25836)
- A connection error would occur when connecting to a MySQL server with certain character sets. Some collations/character sets reported as "unknown" (specifically `cias` variants of existing character sets), and inability to override the detected server character set. (Bug#23645)
- Using `setFetchSize()` breaks prepared `SHOW` and other commands. (Bug#24360)
- Using `DATETIME` columns would result in time shifts when `useServerPrepStmts` was true. The reason was due to different behavior when using client-side compared to server-side prepared statements and the `useJDBCCompliantTimezoneShift` option. This is now fixed if moving from server-side prepared statements to client-side prepared statements by setting `useSSPSCompatibleTimezoneShift` to `true`, as the driver can't tell if this is a new deployment that never used server-side prepared statements, or if it is an existing deployment that is switching to client-side prepared statements from server-side prepared statements. (Bug#24344)
- Inconsistency between `getSchemas` and `INFORMATION_SCHEMA`. (Bug#23304)
- When using the `rewriteBatchedStatements` connection option with `PreparedStatement.executeBatch()` an internal memory leak would occur. (Bug#25073)
- Fixed issue where field-level for metadata from `DatabaseMetaData` when using `INFORMATION_SCHEMA` didn't have references to current connections, sometimes leading to Null Pointer Exceptions (NPEs) when introspecting them via `ResultSet` -

MetaData.

- Connector/J now returns a better error message when server doesn't return enough information to determine stored procedure/function parameter types. ([Bug#24065](#))
- When using server-side prepared statements and timestamp columns, value would be incorrectly populated (with nanoseconds, not microseconds). ([Bug#21438](#))
- Timer instance used for `Statement.setQueryTimeout()` created per-connection, rather than per-VM, causing memory leak. ([Bug#25514](#))
- Results sets from `UPDATE` statements that are part of multi-statement queries would cause an `SQLException` error, "Result is from UPDATE". ([Bug#25009](#))
- `StringUtils.indexOfIgnoreCaseRespectQuotes()` isn't case-insensitive on the first character of the target. This bug also affected `rewriteBatchedStatements` functionality when prepared statements did not use uppercase for the `VALUES` clause. ([Bug#25047](#))
- Some exceptions thrown out of `StandardSocketFactory` were needlessly wrapped, obscuring their true cause, especially when using socket timeouts. ([Bug#21480](#))
- `DatabaseMetaData.getSchemas()` doesn't return a `TABLE_CATALOG` column. ([Bug#23303](#))
- `EscapeProcessor` gets confused by multiple backslashes. We now push the responsibility of syntax errors back on to the server for most escape sequences. ([Bug#25399](#))
- `INOUT` parameters in `CallableStatements` get doubly-escaped. ([Bug#25379](#))
- Connection property `socketFactory` wasn't exposed via correctly named mutator/accessor, causing data source implementations that use JavaBean naming conventions to set properties to fail to set the property (and in the case of SJAS, fail silently when trying to set this parameter). ([Bug#26326](#))
- `ParameterMetaData` throws `NullPointerException` when prepared SQL has a syntax error. Added `generateSimpleParameterMetadata` configuration property, which when set to `true` will generate metadata reflecting `VARCHAR` for every parameter (the default is `false`, which will cause an exception to be thrown if no parameter metadata for the statement is actually available). ([Bug#21267](#))

Other changes:

- Performance enhancement of initial character set configuration, driver will only send commands required to configure connection character set session variables if the current values on the server do not match what is required.
- Re-worked stored procedure parameter parser to be more robust. Driver no longer requires `BEGIN` in stored procedure definition, but does have requirement that if a stored function begins with a label directly after the "returns" clause, that the label is not a quoted identifier.
- Reverted back to internal character conversion routines for single-byte character sets, as the ones internal to the JVM are using much more CPU time than our internal implementation.
- Changed cached result set metadata (when using `cacheResultSetMetadata=true`) to be cached per-connection rather than per-statement as previously implemented.
- Use a `java.util.TreeMap` to map column names to ordinal indexes for `ResultSet.findColumn()` instead of a `HashMap`. This allows us to have case-insensitive lookups (required by the JDBC specification) without resorting to the many transient object instances needed to support this requirement with a normal `HashMap` with either case-adjusted keys, or case-insensitive keys. (In the worst case scenario for lookups of a 1000 column result set, `TreeMaps` are about half as fast wall-clock time as a `HashMap`, however in normal applications their use gives many orders of magnitude reduction in transient object instance creation which pays off later for CPU usage in garbage collection).
- Avoid static synchronized code in JVM class libraries for dealing with default timezones.
- Fixed cases where `ServerPreparedStatements` weren't using cached metadata when `cacheResultSetMetadata=true` was used.

- When using cached metadata, skip field-level metadata packets coming from the server, rather than reading them and discarding them without creating `com.mysql.jdbc.Field` instances.
- Throw exceptions encountered during timeout to thread calling `Statement.execute*`(), rather than `RuntimeException`.
- Take `localSocketAddress` property into account when creating instances of `CommunicationsException` when the underlying exception is a `java.net.BindException`, so that a friendlier error message is given with a little internal diagnostics.
- Fixed some Null Pointer Exceptions (NPEs) when cached metadata was used with `UpdatableResultSets`.
- When extracting foreign key information from `SHOW CREATE TABLE` in `DatabaseMetaData`, ignore exceptions relating to tables being missing (which could happen for cross-reference or imported-key requests, as the list of tables is generated first, then iterated).

C.5.2.5. Changes in MySQL Connector/J 5.0.4 (20 October 2006)

Bugs fixed:

- Column names don't match metadata in cases where server doesn't return original column names (column functions) thus breaking compatibility with applications that expect 1-1 mappings between `findColumn()` and `rsmd.getColumnNames()`, usually manifests itself as "Can't find column (")" exceptions. ([Bug#21379](#))
- When using `information_schema` for metadata, `COLUMN_SIZE` for `getColumns()` is not clamped to range of `java.lang.Integer` as is the case when not using `information_schema`, thus leading to a truncation exception that isn't present when not using `information_schema`. ([Bug#21544](#))
- Newlines causing whitespace to span confuse procedure parser when getting parameter metadata for stored procedures. ([Bug#22024](#))
- Driver was using milliseconds for `Statement.setQueryTimeout()` when specification says argument is to be in seconds. ([Bug#22359](#))
- Workaround for server crash when calling stored procedures via a server-side prepared statement (driver now detects prepare(stored procedure) and substitutes client-side prepared statement). ([Bug#22297](#))
- Added new `_ci` collations to `CharsetMapping` - `utf8_unicode_ci` not working. ([Bug#22456](#))
- Driver issues truncation on write exception when it shouldn't (due to sending big decimal incorrectly to server with server-side prepared statement). ([Bug#22290](#))
- `DBMD.getColumns()` does not return expected `COLUMN_SIZE` for the SET type, now returns length of largest possible set disregarding whitespace or the "," delimiters to be consistent with the ODBC driver. ([Bug#22613](#))

Other changes:

- Fixed configuration property `jdbcCompliantTruncation` was not being used for reads of result set values.
- Driver now supports `{call sp}` (without "()" if procedure has no arguments).
- Driver now sends numeric 1 or 0 for client-prepared statement `setBoolean()` calls instead of '1' or '0'.
- `DatabaseMetaData` correctly reports `true` for `supportsCatalog*`() methods.

C.5.2.6. Changes in MySQL Connector/J 5.0.3 (26 July 2006)

- Fixed `Statement.cancel()` causes `NullPointerException` if underlying connection has been closed due to server failure. ([Bug#20650](#))
- Added configuration option `noAccessToProcedureBodies` which will cause the driver to create basic parameter metadata for `CallableStatements` when the user does not have access to procedure bodies via `SHOW CREATE PROCEDURE` or selecting from `mysql.proc` instead of throwing an exception. The default value for this option is `false`

Bugs fixed:

- If the connection to the server has been closed due to a server failure, then the cleanup process will call `Statement.cancel()`, triggering a `NullPointerException`, even though there is no active connection. (Bug#20650)

C.5.2.7. Changes in MySQL Connector/J 5.0.2-beta (11 July 2006)

- Fixed can't use `XAConnection` for local transactions when no global transaction is in progress. (Bug#17401)
- Fixed driver fails on non-ASCII platforms. The driver was assuming that the platform character set would be a superset of MySQL's `latin1` when doing the handshake for authentication, and when reading error messages. We now use Cp1252 for all strings sent to the server during the handshake phase, and a hard-coded mapping of the `language` system variable to the character set that is used for error messages. (Bug#18086)
- Fixed `ConnectionProperties` (and thus some subclasses) are not serializable, even though some J2EE containers expect them to be. (Bug#19169)
- Fixed `MySQLValidConnectionChecker` for JBoss doesn't work with `MySQLXADataSources`. (Bug#20242)
- Better caching of character set converters (per-connection) to remove a bottleneck for multibyte character sets.
- Added connection/datasource property `pinGlobalTxToPhysicalConnection` (defaults to `false`). When set to `true`, when using `XAConnections`, the driver ensures that operations on a given XID are always routed to the same physical connection. This allows the `XAConnection` to support `XA START ... JOIN` after `XA END` has been called, and is also a workaround for transaction managers that don't maintain thread affinity for a global transaction (most either always maintain thread affinity, or have it as a configuration option).
- `MysqlXaConnection.recover(int flags)` now allows combinations of `XAResource.TMSTARTRSCAN` and `TMENDRSCAN`. To simulate the “scanning” nature of the interface, we return all prepared XIDs for `TMSTARTRSCAN`, and no new XIDs for calls with `TMNOFLAGS`, or `TMENDRSCAN` when not in combination with `TMSTARTRSCAN`. This change was made for API compliance, as well as integration with IBM WebSphere's transaction manager.

C.5.2.8. Changes in MySQL Connector/J 5.0.1-beta (Not Released)

Not released due to a packaging error

C.5.2.9. Changes in MySQL Connector/J 5.0.0-beta (22 December 2005)

- `XADataSource` implemented (ported from 3.2 branch which won't be released as a product). Use `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` as your datasource class name in your application server to utilize XA transactions in MySQL-5.0.10 and newer.
- `PreparedStatement.setString()` didn't work correctly when `sql_mode` on server contained `NO_BACKSLASH_ESCAPES` and no characters that needed escaping were present in the string.
- Attempt detection of the MySQL type `BINARY` (it's an alias, so this isn't always reliable), and use the `java.sql.Types.BINARY` type mapping for it.
- Moved `-bin-g.jar` file into separate `debug` subdirectory to avoid confusion.
- Don't allow `.setAutoCommit(true)`, or `.commit()` or `.rollback()` on an XA-managed connection as per the JDBC specification.
- If the connection `useTimezone` is set to `true`, then also respect time zone conversions in escape-processed string literals (for example, `"{ts ...}"` and `"{t ...}"`).
- Return original column name for `RSMD.getColumnNames()` if the column was aliased, alias name for `.getColumnLabel()` (if aliased), and original table name for `.getTableName()`. Note this only works for MySQL-4.1 and newer, as older servers don't make this information available to clients.

- Setting `useJDBCCompliantTimezoneShift=true` (it's not the default) causes the driver to use GMT for *all* `TIMESTAMP/DATETIME` time zones, and the current VM time zone for any other type that refers to time zones. This feature can not be used when `useTimezone=true` to convert between server and client time zones.
- Add one level of indirection of internal representation of `CallableStatement` parameter metadata to avoid class not found issues on JDK-1.3 for `ParameterMetadata` interface (which doesn't exist prior to JDBC-3.0).
- Added unit tests for `XADataSource`, as well as friendlier exceptions for XA failures compared to the "stock" `XAException` (which has no messages).
- Idle timeouts cause `XAConnections` to whine about rolling themselves back. ([Bug#14729](#))
- Added support for Connector/MXJ integration via url subprotocol `jdbc:mysql:mxj://...`
- Moved all `SQLException` constructor usage to a factory in `SQLException` (ground-work for JDBC-4.0 `SQLState`-based exception classes).
- Removed Java5-specific calls to `BigDecimal` constructor (when result set value is `'', (int)0` was being used as an argument indirectly via method return value. This signature doesn't exist prior to Java5.)
- Added service-provider entry to `META-INF/services/java.sql.Driver` for JDBC-4.0 support.
- Return "[VAR]BINARY" for `RSMD.getColumnTypeName()` when that is actually the type, and it can be distinguished (MySQL-4.1 and newer).
- When fix for [Bug#14562](#) was merged from 3.1.12, added functionality for `CallableStatement`'s parameter metadata to return correct information for `.getParameterClassName()`.
- Fuller synchronization of `Connection` to avoid deadlocks when using multithreaded frameworks that multithread a single connection (usually not recommended, but the JDBC spec allows it anyways), part of fix to [Bug#14972](#).
- Implementation of `Statement.cancel()` and `Statement.setQueryTimeout()`. Both require MySQL-5.0.0 or newer server, require a separate connection to issue the `KILL QUERY` statement, and in the case of `setQueryTimeout()` creates an additional thread to handle the timeout functionality.

Note: Failures to cancel the statement for `setQueryTimeout()` may manifest themselves as `RuntimeExceptions` rather than failing silently, as there is currently no way to unblock the thread that is executing the query being cancelled due to timeout expiration and have it throw the exception instead.

C.5.3. Changes in MySQL Connector/J 3.1.x

C.5.3.1. Changes in MySQL Connector/J 3.1.15 (Not yet released)

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is `false` (that is, Connector/J does not use server-side prepared statements).

Bugs fixed:

- Specifying `US-ASCII` as the character set in a connection to a MySQL 4.1 or newer server does not map correctly. ([Bug#24840](#))

C.5.3.2. Changes in MySQL Connector/J 3.1.14 (10-19-2006)

- Fixed updatable result set throws `ClassCastException` when there is row data and `moveToInsertRow()` is called. ([Bug#20479](#))
- Fixed Updatable result set that contains a `BIT` column fails when server-side prepared statements are used. ([Bug#20485](#))

- Fixed memory leak with profileSQL=true. ([Bug#16987](#))
- Connection fails to localhost when using timeout and IPv6 is configured. ([Bug#19726](#))
- Fixed NullPointerException in MysqlDataSourceFactory due to Reference containing RefAddr with null content. ([Bug#16791](#))
- Fixed ResultSet.getShort() for UNSIGNED TINYINT returns incorrect values when using server-side prepared statements. ([Bug#20306](#))
- Fixed can't pool server-side prepared statements, exception raised when re-using them. ([Bug#20687](#))
- ResultSet.getSomeInteger() doesn't work for BIT(>1). ([Bug#21062](#))
- ResultSet.getFloatFromString() can't retrieve values near Float.MIN/MAX_VALUE. ([Bug#18880](#))
- Escape of quotes in client-side prepared statements parsing not respected. Patch covers more than bug report, including NO_BACKSLASH_ESCAPES being set, and stacked quote characters forms of escaping (that is, " or ""). ([Bug#20888](#))
- ReplicationDriver does not always round-robin load balance depending on URL used for slaves list. ([Bug#19993](#))
- Fixed calling toString() on ResultSetMetaData for driver-generated (that is, from DatabaseMetaData method calls, or from getGeneratedKeys()) result sets would raise a NullPointerException.
- DDriver throws NPE when tracing prepared statements that have been closed (in asSQL()). ([Bug#21207](#))
- Removed logger autodetection altogether, must now specify logger explicitly if you want to use a logger other than one that logs to STDERR.
- Driver issues truncation on write exception when it shouldn't (due to sending big decimal incorrectly to server with server-side prepared statement). ([Bug#22290](#))
- Driver now sends numeric 1 or 0 for client-prepared statement setBoolean() calls instead of '1' or '0'.
- Fixed bug where driver would not advance to next host if roundRobinLoadBalance=true and the last host in the list is down.
- Fixed [Bug#18258](#) - DatabaseMetaData.getTables(), columns() with bad catalog parameter threw exception rather than return empty result set (as required by spec).
- Check and store value for continueBatchOnError property in constructor of Statements, rather than when executing batches, so that Connections closed out from underneath statements don't cause NullPointerExceptions when it's required to check this property.
- Fixed bug when calling stored functions, where parameters weren't numbered correctly (first parameter is now the return value, subsequent parameters if specified start at index "2").

C.5.3.3. Changes in MySQL Connector/J 3.1.13 (26 May 2006)

- **INOUT** parameter does not store **IN** value. ([Bug#15464](#))
- Exception thrown for new decimal type when using updatable result sets. ([Bug#14609](#))
- No "dos" character set in MySQL > 4.1.0. ([Bug#15544](#))
- `PreparedStatement.setObject()` serializes **BigInteger** as object, rather than sending as numeric value (and is thus not complementary to `.getObject()` on an **UNSIGNED LONG** type). ([Bug#15383](#))
- `ResultSet.getShort()` for **UNSIGNED TINYINT** returned wrong values. ([Bug#11874](#))
- **lib-nodist** directory missing from package breaks out-of-box build. ([Bug#15676](#))
- `DBMD.getColumns()` returns wrong type for **BIT**. ([Bug#15854](#))
- Fixed issue where driver was unable to initialize character set mapping tables. Removed reliance on `.properties` files to hold this information, as it turns out to be too problematic to code around class loader hierarchies that change depending on how an application is deployed. Moved information back into the `CharsetMapping` class. ([Bug#14938](#))

- Fixed updatable result set doesn't return `AUTO_INCREMENT` values for `insertRow()` when multiple column primary keys are used. (the driver was checking for the existence of single-column primary keys and an autoincrement value > 0 instead of a straightforward `isAutoIncrement()` check). ([Bug#16841](#))
- Fixed `Statement.getGeneratedKeys()` throws `NullPointerException` when no query has been processed. ([Bug#17099](#))
- Fixed driver trying to call methods that don't exist on older and newer versions of Log4j. The fix is not trying to auto-detect presence of log4j, too many different incompatible versions out there in the wild to do this reliably. ([Bug#13469](#))

If you relied on autodetection before, you will need to add "logger=com.mysql.jdbc.log.Log4JLogger" to your JDBC URL to enable Log4J usage, or alternatively use the new "CommonsLogger" class to take care of this.
- Added support for Apache Commons logging, use "com.mysql.jdbc.log.CommonsLogger" as the value for the "logger" configuration property.
- LogFactory now prepends "com.mysql.jdbc.log" to log class name if it can't be found as-specified. This allows you to use "short names" for the built-in log factories, for example "logger=CommonsLogger" instead of "logger=com.mysql.jdbc.log.CommonsLogger".
- Fixed issue with `ReplicationConnection` incorrectly copying state, doesn't transfer connection context correctly when transitioning between the same read-only states. ([Bug#15570](#))
- Fixed issue where server-side prepared statements don't cause truncation exceptions to be thrown when truncation happens. ([Bug#18041](#))
- Added performance feature, re-writing of batched executes for `Statement.executeBatch()` (for all DML statements) and `PreparedStatement.executeBatch()` (for INSERTs with VALUE clauses only). Enable by using "rewriteBatchedStatements=true" in your JDBC URL.
- Fixed `CallableStatement.registerOutParameter()` not working when some parameters pre-populated. Still waiting for feedback from JDBC experts group to determine what correct parameter count from `getMetaData()` should be, however. ([Bug#17898](#))
- Fixed calling `clearParameters()` on a closed prepared statement causes NPE. ([Bug#17587](#))
- Map "latin1" on MySQL server to CP1252 for MySQL > 4.1.0.
- Added additional accessor and mutator methods on `ConnectionProperties` so that `DataSource` users can use same naming as regular URL properties.
- Fixed data truncation and `getWarnings()` only returns last warning in set. ([Bug#18740](#))
- Improved performance of retrieving `BigDecimal`, `Time`, `Timestamp` and `Date` values from server-side prepared statements by creating fewer short-lived instances of `Strings` when the native type is not an exact match for the requested type. ([Bug#18496](#))
- Fixed aliased column names where length of name > 251 are corrupted. ([Bug#18554](#))
- Fixed `ResultSet.wasNull()` not always reset correctly for booleans when done via conversion for server-side prepared statements. ([Bug#17450](#))
- Fixed invalid classname returned for `ResultSetMetaData.getColumnClassName()` for `BIGINT` type. ([Bug#19282](#))
- Fixed case where driver wasn't reading server status correctly when fetching server-side prepared statement rows, which in some cases could cause warning counts to be off, or multiple result sets to not be read off the wire.
- Driver now aware of fix for `BIT` type metadata that went into MySQL-5.0.21 for server not reporting length consistently ([Bug#13601](#)).
- Fixed `PreparedStatement.setObject(int, Object, int)` doesn't respect scale of `BigDecimal`s. ([Bug#19615](#))
- Fixed `ResultSet.wasNull()` returns incorrect value when extracting native string from server-side prepared statement generated result set. ([Bug#19282](#))

C.5.3.4. Changes in MySQL Connector/J 3.1.12 (30 November 2005)

- Fixed client-side prepared statement bug with embedded `?` characters inside quoted identifiers (it was recognized as a placeholder, when it was not).
- Don't allow `executeBatch()` for `CallableStatements` with registered `OUT/INOUT` parameters (JDBC compliance).
- Fall back to platform-encoding for `URLDecoder.decode()` when parsing driver URL properties if the platform doesn't have a two-argument version of this method.
- Java type conversion may be incorrect for `MEDIUMINT`. ([Bug#14562](#))
- Added configuration property `useGmtMillisForDatetimes` which when set to `true` causes `ResultSet.getDate()`, `.getTimestamp()` to return correct millis-since GMT when `.getTime()` is called on the return value (currently default is `false` for legacy behavior).
- Fixed `DatabaseMetaData.stores*Identifiers()`:
 - If `lower_case_table_names=0` (on server):
 - `storesLowerCaseIdentifiers()` returns `false`
 - `storesLowerCaseQuotedIdentifiers()` returns `false`
 - `storesMixedCaseIdentifiers()` returns `true`
 - `storesMixedCaseQuotedIdentifiers()` returns `true`
 - `storesUpperCaseIdentifiers()` returns `false`
 - `storesUpperCaseQuotedIdentifiers()` returns `true`
 - If `lower_case_table_names=1` (on server):
 - `storesLowerCaseIdentifiers()` returns `true`
 - `storesLowerCaseQuotedIdentifiers()` returns `true`
 - `storesMixedCaseIdentifiers()` returns `false`
 - `storesMixedCaseQuotedIdentifiers()` returns `false`
 - `storesUpperCaseIdentifiers()` returns `false`
 - `storesUpperCaseQuotedIdentifiers()` returns `true`
- `DatabaseMetaData.getColumns()` doesn't return `TABLE_NAME` correctly. ([Bug#14815](#))
- Escape processor replaces quote character in quoted string with string delimiter. ([Bug#14909](#))
- OpenOffice expects `DBMD.supportsIntegrityEnhancementFacility()` to return `true` if foreign keys are supported by the datasource, even though this method also covers support for check constraints, which MySQL *doesn't* have. Setting the configuration property `overrideSupportsIntegrityEnhancementFacility` to `true` causes the driver to return `true` for this method. ([Bug#12975](#))
- Added `com.mysql.jdbc.testsuite.url.default` system property to set default JDBC url for testsuite (to speed up bug resolution when I'm working in Eclipse).
- Unable to initialize character set mapping tables (due to J2EE classloader differences). ([Bug#14938](#))
- Deadlock while closing server-side prepared statements from multiple threads sharing one connection. ([Bug#14972](#))
- `logSlowQueries` should give better info. ([Bug#12230](#))
- Extraneous sleep on `autoReconnect`. ([Bug#13775](#))
- Driver incorrectly closes streams passed as arguments to `PreparedStatements`. Reverts to legacy behavior by setting the JDBC configuration property `autoClosePstmtStreams` to `true` (also included in the 3-0-Compat configuration "bundle"). ([Bug#15024](#))

- `maxQuerySizeToLog` is not respected. Added logging of bound values for `execute()` phase of server-side prepared statements when `profileSQL=true` as well. (Bug#13048)
- Usage advisor complains about unreferenced columns, even though they've been referenced. (Bug#15065)
- Don't increase timeout for failover/reconnect. (Bug#6577)
- Process escape tokens in `Connection.prepareStatement(...)`. (Bug#15141) You can disable this behavior by setting the JDBC URL configuration property `processEscapeCodesForPrepStmts` to `false`.
- Reconnect during middle of `executeBatch()` should not occur if `autoReconnect` is enabled. (Bug#13255)

C.5.3.5. Changes in MySQL Connector/J 3.1.11-stable (07 October 2005)

- Spurious `!` on console when character encoding is `utf8`. (Bug#11629)
- Fixed statements generated for testcases missing `;` for “plain” statements.
- Incorrect generation of testcase scripts for server-side prepared statements. (Bug#11663)
- Fixed regression caused by fix for Bug#11552 that caused driver to return incorrect values for unsigned integers when those integers were within the range of the positive signed type.
- Moved source code to Subversion repository.
- Escape tokenizer doesn't respect stacked single quotes for escapes. (Bug#11797)
- `GEOMETRY` type not recognized when using server-side prepared statements.
- `ReplicationConnection` won't switch to slave, throws “Catalog can't be null” exception. (Bug#11879)
- Properties shared between master and slave with replication connection. (Bug#12218)
- `Statement.getWarnings()` fails with NPE if statement has been closed. (Bug#10630)
- Only get `char[]` from SQL in `PreparedStatement.ParseInfo()` when needed.
- Geometry types not handled with server-side prepared statements. (Bug#12104)
- `StringUtils.getBytes()` doesn't work when using multi-byte character encodings and a length in *characters* is specified. (Bug#11614)
- `Pstmt.setObject(..., Types.BOOLEAN)` throws exception. (Bug#11798)
- `maxPerformance.properties` mis-spells “elideSetAutoCommits”. (Bug#11976)
- `DBMD.storesLower/Mixed/UpperIdentifiers()` reports incorrect values for servers deployed on Windows. (Bug#11575)
- `ResultSet.moveToCurrentRow()` fails to work when preceded by a call to `ResultSet.moveToInsertRow()`. (Bug#11190)
- `VARBINARY` data corrupted when using server-side prepared statements and `.setBytes()`. (Bug#11115)
- `explainSlowQueries` hangs with server-side prepared statements. (Bug#12229)
- Escape processor didn't honor strings demarcated with double quotes. (Bug#11498)
- Lifted restriction of changing streaming parameters with server-side prepared statements. As long as `all` streaming parameters were set before execution, `.clearParameters()` does not have to be called. (due to limitation of client/server protocol, prepared statements can not reset *individual* stream data on the server side).
- Reworked `Field` class, `*Buffer`, and `MysqlIO` to be aware of field lengths > `Integer.MAX_VALUE`.

- Updated `DBMD.supportsCorrelatedQueries()` to return `true` for versions > 4.1, `supportsGroupByUnrelated()` to return `true` and `getResultSetHoldability()` to return `HOLD_CURSORS_OVER_COMMIT`.
- Handling of catalog argument in `DatabaseMetaData.getIndexInfo()`, which also means changes to the following methods in `DatabaseMetaData`: ([Bug#12541](#))
 - `getBestRowIdentifier()`
 - `getColumns()`
 - `getCrossReference()`
 - `getExportedKeys()`
 - `getImportedKeys()`
 - `getIndexInfo()`
 - `getPrimaryKeys()`
 - `getProcedures()` (and thus indirectly `getProcedureColumns()`)
 - `getTables()`

The `catalog` argument in all of these methods now behaves in the following way:

- Specifying `NULL` means that catalog will not be used to filter the results (thus all databases will be searched), unless you've set `nullCatalogMeansCurrent=true` in your JDBC URL properties.
- Specifying `" "` means “current” catalog, even though this isn't quite JDBC spec compliant, it's there for legacy users.
- Specifying a catalog works as stated in the API docs.
- Made `Connection.clientPrepare()` available from “wrapped” connections in the `jdbc2.optional` package (connections built by `ConnectionPoolDataSource` instances).
- Added `Connection.isMasterConnection()` for clients to be able to determine if a multi-host master/slave connection is connected to the first host in the list.
- Tokenizer for `=` in URL properties was causing `sessionVariables=...` to be parameterized incorrectly. ([Bug#12753](#))
- Foreign key information that is quoted is parsed incorrectly when `DatabaseMetaData` methods use that information. ([Bug#11781](#))
- The `sendBlobChunkSize` property is now clamped to `max_allowed_packet` with consideration of stream buffer size and packet headers to avoid `PacketTooBigExceptions` when `max_allowed_packet` is similar in size to the default `send-BlobChunkSize` which is 1M.
- `CallableStatement.clearParameters()` now clears resources associated with `INOUT/OUTPUT` parameters as well as `INPUT` parameters.
- `Connection.prepareCall()` is database name case-sensitive (on Windows systems). ([Bug#12417](#))
- `cp1251` incorrectly mapped to `win1251` for servers newer than 4.0.x. ([Bug#12752](#))
- `java.sql.Types.OTHER` returned for `BINARY` and `VARBINARY` columns when using `Database-MetaData.getColumns()`. ([Bug#12970](#))
- `ServerPreparedStatement.getBinding()` now checks if the statement is closed before attempting to reference the list of parameter bindings, to avoid throwing a `NullPointerException`.
- `ResultSetMetaData` from `Statement.getGeneratedKeys()` caused a `NullPointerException` to be thrown whenever a method that required a connection reference was called. ([Bug#13277](#))
- Backport of `Field` class, `ResultSetMetaData.getColumnClassName()`, and `ResultSet.getObject(int)` changes from 5.0 branch to fix behavior surrounding `VARCHAR BINARY/VARBINARY` and related types.

- Fixed `NullPointerException` when converting `catalog` parameter in many `DatabaseMetaDataMethods` to `byte[]`s (for the result set) when the parameter is `null`. (`null` isn't technically allowed by the JDBC specification, but we've historically allowed it).
- Backport of `VAR[BINARY|CHAR] [BINARY]` types detection from 5.0 branch.
- Read response in `MysqlIO.sendFileToServer()`, even if the local file can't be opened, otherwise next query issued will fail, because it's reading the response to the empty `LOAD DATA INFILE` packet sent to the server.
- Workaround for [Bug#13374](#): `ResultSet.getStatement()` on closed result set returns `NULL` (as per JDBC 4.0 spec, but not backward-compatible). Set the connection property `retainStatementAfterResultSetClose` to `true` to be able to retrieve a `ResultSet`'s statement after the `ResultSet` has been closed via `.getStatement()` (the default is `false`, to be JDBC-compliant and to reduce the chance that code using JDBC leaks `Statement` instances).
- URL configuration parameters don't allow `'&'` or `'='` in their values. The JDBC driver now parses configuration parameters as if they are encoded using the application/x-www-form-urlencoded format as specified by `java.net.URLDecoder` (<http://java.sun.com/j2se/1.5.0/docs/api/java/net/URLDecoder.html>). ([Bug#13453](#))

If the `'%'` character is present in a configuration property, it must now be represented as `%25`, which is the encoded form of `'%'` when using application/x-www-form-urlencoded encoding.
- The configuration property `sessionVariables` now allows you to specify variables that start with the `'@'` sign.
- When `gatherPerfMetrics` is enabled for servers older than 4.1.0, a `NullPointerException` is thrown from the constructor of `ResultSet` if the query doesn't use any tables. ([Bug#13043](#))

C.5.3.6. Changes in MySQL Connector/J 3.1.10-stable (23 June 2005)

- Fixed connecting without a database specified raised an exception in `MysqlIO.changeDatabaseTo()`.
- Initial implementation of `ParameterMetadata` for `PreparedStatement.getParameterMetadata()`. Only works fully for `CallableStatements`, as current server-side prepared statements return every parameter as a `VARCHAR` type.

C.5.3.7. Changes in MySQL Connector/J 3.1.9-stable (22 June 2005)

- Overhaul of character set configuration, everything now lives in a properties file.
- Driver now correctly uses CP932 if available on the server for Windows-31J, CP932 and MS932 java encoding names, otherwise it resorts to SJIS, which is only a close approximation. Currently only MySQL-5.0.3 and newer (and MySQL-4.1.12 or .13, depending on when the character set gets backported) can reliably support any variant of CP932.
- `com.mysql.jdbc.PreparedStatement.ParseInfo` does unnecessary call to `toCharArray()`. ([Bug#9064](#))
- Memory leak in `ServerPreparedStatement` if `serverPrepare()` fails. ([Bug#10144](#))
- Actually write manifest file to correct place so it ends up in the binary jar file.
- Added `createDatabaseIfNotExist` property (default is `false`), which will cause the driver to ask the server to create the database specified in the URL if it doesn't exist. You must have the appropriate privileges for database creation for this to work.
- Unsigned `SMALLINT` treated as signed for `ResultSet.getInt()`, fixed all cases for `UNSIGNED` integer values and server-side prepared statements, as well as `ResultSet.getObject()` for `UNSIGNED TINYINT`. ([Bug#10156](#))
- Double quotes not recognized when parsing client-side prepared statements. ([Bug#10155](#))
- Made `enableStreamingResults()` visible on `com.mysql.jdbc.jdbc2.optional.StatementWrapper`.
- Made `ServerPreparedStatement.asSql()` work correctly so auto-explain functionality would work with server-side prepared statements.
- Made JDBC2-compliant wrappers public in order to allow access to vendor extensions.

- Cleaned up logging of profiler events, moved code to dump a profiler event as a string to `com.mysql.jdbc.log.LogUtils` so that third parties can use it.
- `DatabaseMetaData.supportsMultipleOpenResults()` now returns `true`. The driver has supported this for some time, DBMD just missed that fact.
- Driver doesn't support `{?=CALL( ... )}` for calling stored functions. This involved adding support for function retrieval to `DatabaseMetaData.getProcedures()` and `getProcedureColumns()` as well. (Bug#10310)
- `SQLException` thrown when retrieving `YEAR(2)` with `ResultSet.getString()`. The driver will now always treat `YEAR` types as `java.sql.Date`s and return the correct values for `getString()`. Alternatively, the `yearIsDateType` connection property can be set to `false` and the values will be treated as `SHORT`s. (Bug#10485)
- The datatype returned for `TINYINT(1)` columns when `tinyInt1isBit=true` (the default) can be switched between `Types.BOOLEAN` and `Types.BIT` using the new configuration property `transformedBitIsBoolean`, which defaults to `false`. If set to `false` (the default), `DatabaseMetaData.getColumns()` and `ResultSetMetaData.getColumnType()` will return `Types.BOOLEAN` for `TINYINT(1)` columns. If `true`, `Types.BOOLEAN` will be returned instead. Regardless of this configuration property, if `tinyInt1isBit` is enabled, columns with the type `TINYINT(1)` will be returned as `java.lang.Boolean` instances from `ResultSet.getObject(...)`, and `ResultSetMetaData.getColumnClassName()` will return `java.lang.Boolean`.
- `SQLException` is thrown when using property `characterSetResults` with `cp932` or `eucjpms`. (Bug#10496)
- Reorganized directory layout. Sources now are in `src` folder. Don't pollute parent directory when building, now output goes to `./build`, distribution goes to `./dist`.
- Added support/bug hunting feature that generates `.sql` test scripts to `STDERR` when `autoGenerateTestcaseScript` is set to `true`.
- 0-length streams not sent to server when using server-side prepared statements. (Bug#10850)
- Setting `cachePrepStmts=true` now causes the `Connection` to also cache the check the driver performs to determine if a prepared statement can be server-side or not, as well as caches server-side prepared statements for the lifetime of a connection. As before, the `prepStmtCacheSize` parameter controls the size of these caches.
- Try to handle `OutOfMemoryErrors` more gracefully. Although not much can be done, they will in most cases close the connection they happened on so that further operations don't run into a connection in some unknown state. When an OOM has happened, any further operations on the connection will fail with a "Connection closed" exception that will also list the OOM exception as the reason for the implicit connection close event.
- Don't send `COM_RESET_STMT` for each execution of a server-side prepared statement if it isn't required.
- Driver detects if you're running MySQL-5.0.7 or later, and does not scan for `LIMIT ?[, ?]` in statements being prepared, as the server supports those types of queries now.
- `VARBINARY` data corrupted when using server-side prepared statements and `ResultSet.getBytes()`. (Bug#11115)
- `Connection.setCatalog()` is now aware of the `useLocalSessionState` configuration property, which when set to `true` will prevent the driver from sending `USE ...` to the server if the requested catalog is the same as the current catalog.
- Added the following configuration bundles, use one or many via the `useConfigs` configuration property:
 - `maxPerformance` — maximum performance without being reckless
 - `solarisMaxPerformance` — maximum performance for Solaris, avoids syscalls where it can
 - `3-0-Compat` — Compatibility with Connector/J 3.0.x functionality
- Added `maintainTimeStats` configuration property (defaults to `true`), which tells the driver whether or not to keep track of the last query time and the last successful packet sent to the server's time. If set to `false`, removes two syscalls per query.
- `autoReconnect` ping causes exception on connection startup. (Bug#11259)
- Connector/J dumping query into `SQLException` twice. (Bug#11360)
- Fixed `PreparedStatement.setClob()` not accepting `null` as a parameter.

- Production package doesn't include JBoss integration classes. ([Bug#11411](#))
- Removed nonsensical “costly type conversion” warnings when using usage advisor.

C.5.3.8. Changes in MySQL Connector/J 3.1.8-stable (14 April 2005)

- Fixed `DatabaseMetaData.getTables()` returning views when they were not asked for as one of the requested table types.
- Added support for new precision-math `DECIMAL` type in MySQL 5.0.3 and up.
- Fixed `ResultSet.getTime()` on a `NULL` value for server-side prepared statements throws NPE.
- Made `Connection.ping()` a public method.
- `DATE_FORMAT()` queries returned as `BLOBs` from `getObject()`. ([Bug#8868](#))
- `ServerPreparedStatement` now correctly “stream” `BLOB/CLOB` data to the server. You can configure the threshold chunk size using the JDBC URL property `blobSendChunkSize` (the default is 1MB).
- `BlobFromLocator` now uses correct identifier quoting when generating prepared statements.
- Server-side session variables can be preset at connection time by passing them as a comma-delimited list for the connection property `sessionVariables`.
- Fixed regression in `ping()` for users using `autoReconnect=true`.
- `PreparedStatement.addBatch()` doesn't work with server-side prepared statements and streaming `BINARY` data. ([Bug#9040](#))
- `DBMD.supportsMixedCase*Identifiers()` returns wrong value on servers running on case-sensitive filesystems. ([Bug#8800](#))
- Cannot use `UTF-8` for `characterSetResults` configuration property. ([Bug#9206](#))
- A continuation of [Bug#8868](#), where functions used in queries that should return non-string types when resolved by temporary tables suddenly become opaque binary strings (work-around for server limitation). Also fixed fields with type of `CHAR(n)` `CHARACTER SET BINARY` to return correct/matching classes for `RSMD.getColumnClassName()` and `ResultSet.getObject()`. ([Bug#9236](#))
- `DBMD.supportsResultSetConcurrency()` not returning `true` for forward-only/read-only result sets (we obviously support this). ([Bug#8792](#))
- `DATA_TYPE` column from `DBMD.getBestRowIdentifier()` causes `ArrayIndexOutOfBoundsException` when accessed (and in fact, didn't return any value). ([Bug#8803](#))
- Check for empty strings (`' '`) when converting `CHAR/VARCHAR` column data to numbers, throw exception if `emptyString-sConvertToZero` configuration property is set to `false` (for backward-compatibility with 3.0, it is now set to `true` by default, but will most likely default to `false` in 3.2).
- `PreparedStatement.getMetaData()` inserts blank row in database under certain conditions when not using server-side prepared statements. ([Bug#9320](#))
- `Connection.canHandleAsPreparedStatement()` now makes “best effort” to distinguish `LIMIT` clauses with placeholders in them from ones without in order to have fewer false positives when generating work-arounds for statements the server cannot currently handle as server-side prepared statements.
- Fixed `build.xml` to not compile `log4j` logging if `log4j` not available.
- Added support for the c3p0 connection pool's (<http://c3p0.sf.net/>) validation/connection checker interface which uses the lightweight `COM_PING` call to the server if available. To use it, configure your c3p0 connection pool's `connectionTesterClassName` property to use `com.mysql.jdbc.integration.c3p0.MySQLConnectionTester`.
- Better detection of `LIMIT` inside/outside of quoted strings so that the driver can more correctly determine whether a prepared statement can be prepared on the server or not.

- Stored procedures with same name in different databases confuse the driver when it tries to determine parameter counts/types. ([Bug#9319](#))
- Added finalizers to `ResultSet` and `Statement` implementations to be JDBC spec-compliant, which requires that if not explicitly closed, these resources should be closed upon garbage collection.
- Stored procedures with `DECIMAL` parameters with storage specifications that contained ‘,’ in them would fail. ([Bug#9682](#))
- `PreparedStatement.setObject(int, Object, int type, int scale)` now uses scale value for `BigDecimal` instances.
- `Statement.getMoreResults()` could throw NPE when existing result set was `.close()`d. ([Bug#9704](#))
- The performance metrics feature now gathers information about number of tables referenced in a SELECT.
- The logging system is now automatically configured. If the value has been set by the user, via the URL property `logger` or the system property `com.mysql.jdbc.logger`, then use that, otherwise, autodetect it using the following steps:
 1. Log4j, if it's available,
 2. Then JDK1.4 logging,
 3. Then fallback to our `STDERR` logging.
- `DBMD.getTables()` shouldn't return tables if views are asked for, even if the database version doesn't support views. ([Bug#9778](#))
- Fixed driver not returning `true` for `-1` when `ResultSet.getBoolean()` was called on result sets returned from server-side prepared statements.
- Added a `Manifest.MF` file with implementation information to the `.jar` file.
- More tests in `Field.isOpaqueBinary()` to distinguish opaque binary (that is, fields with type `CHAR(n)` and `CHARACTER SET BINARY`) from output of various scalar and aggregate functions that return strings.
- Should accept `null` for catalog (meaning use current) in DBMD methods, even though it's not JDBC-compliant for legacy's sake. Disable by setting connection property `nullCatalogMeansCurrent` to `false` (which will be the default value in C/J 3.2.x). ([Bug#9917](#))
- Should accept `null` for name patterns in DBMD (meaning ‘%’), even though it isn't JDBC compliant, for legacy's sake. Disable by setting connection property `nullNamePatternMatchesAll` to `false` (which will be the default value in C/J 3.2.x). ([Bug#9769](#))

C.5.3.9. Changes in MySQL Connector/J 3.1.7-stable (18 February 2005)

- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. ([Bug#7686](#))
- Timestamps converted incorrectly to strings with server-side prepared statements and updatable result sets. ([Bug#7715](#))
- Detect new `sql_mode` variable in string form (it used to be integer) and adjust quoting method for strings appropriately.
- Added `holdResultsOpenOverStatementClose` property (default is `false`), that keeps result sets open over `statement.close()` or new execution on same statement (suggested by Kevin Burton).
- Infinite recursion when “falling back” to master in failover configuration. ([Bug#7952](#))
- Disable multi-statements (if enabled) for MySQL-4.1 versions prior to version 4.1.10 if the query cache is enabled, as the server returns wrong results in this configuration.
- Fixed duplicated code in `configureClientCharset()` that prevented `useOldUTF8Behavior=true` from working properly.
- Removed `dontUnpackBinaryResults` functionality, the driver now always stores results from server-side prepared statements as is from the server and unpacks them on demand.

- Emulated locators corrupt binary data when using server-side prepared statements. (Bug#8096)
- Fixed synchronization issue with `ServerPreparedStatement.serverPrepare()` that could cause deadlocks/crashes if connection was shared between threads.
- By default, the driver now scans SQL you are preparing via all variants of `Connection.prepareStatement()` to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this by passing `emulateUnsupportedPstmts=false` in your JDBC URL. (Bug#4718)
- Remove `_binary` introducer from parameters used as in/out parameters in `CallableStatement`.
- Always return `byte[]`s for output parameters registered as `*BINARY`.
- Send correct value for “boolean” `true` to server for `PreparedStatement.setObject(n, "true", Types.BIT)`.
- Fixed bug with `Connection` not caching statements from `prepareStatement()` when the statement wasn't a server-side prepared statement.
- Choose correct “direction” to apply time adjustments when both client and server are in GMT time zone when using `ResultSet.get(..., cal)` and `PreparedStatement.set(..., cal)`.
- Added `dontTrackOpenResources` option (default is `false`, to be JDBC compliant), which helps with memory use for non-well-behaved apps (that is, applications that don't close `Statement` objects when they should).
- `ResultSet.getString()` doesn't maintain format stored on server, bug fix only enabled when `noDatetimeStringSync` property is set to `true` (the default is `false`). (Bug#8428)
- Fixed NPE in `ResultSet.realClose()` when using usage advisor and result set was already closed.
- `PreparedStatements` not creating streaming result sets. (Bug#8487)
- Don't pass `NULL` to `String.valueOf()` in `ResultSet.getNativeConvertToString()`, as it stringifies it (that is, returns `null`), which is not correct for the method in question.
- `ResultSet.getBigDecimal()` throws exception when rounding would need to occur to set scale. The driver now chooses a rounding mode of “half up” if non-rounding `BigDecimal.setScale()` fails. (Bug#8424)
- Added `useLocalSessionState` configuration property, when set to `true` the JDBC driver trusts that the application is well-behaved and only sets autocommit and transaction isolation levels using the methods provided on `java.sql.Connection`, and therefore can manipulate these values in many cases without incurring round-trips to the database server.
- Added `enableStreamingResults()` to `Statement` for connection pool implementations that check `Statement.setFetchSize()` for specification-compliant values. Call `Statement.setFetchSize(>=0)` to disable the streaming results for that statement.
- Added support for `BIT` type in MySQL-5.0.3. The driver will treat `BIT(1-8)` as the JDBC standard `BIT` type (which maps to `java.lang.Boolean`), as the server does not currently send enough information to determine the size of a bitfield when `< 9` bits are declared. `BIT(>9)` will be treated as `VARBINARY`, and will return `byte[]` when `getObject()` is called.

C.5.3.10. Changes in MySQL Connector/J 3.1.6-stable (23 December 2004)

- Fixed hang on `SocketInputStream.read()` with `Statement.setMaxRows()` and multiple result sets when driver has to truncate result set directly, rather than tacking a `LIMIT n` on the end of it.
- `DBMD.getProcedures()` doesn't respect catalog parameter. (Bug#7026)

C.5.3.11. Changes in MySQL Connector/J 3.1.5-gamma (02 December 2004)

- Fix comparisons made between string constants and dynamic strings that are converted with either `toUpperCase()` or `toLowerCase()` to use `Locale.ENGLISH`, as some locales “override” case rules for English. Also use `StringUtils.indexOfIgnoreCase()` instead of `.toUpperCase().indexOf()`, avoids creating a very short-lived transient

`String` instance.

- Server-side prepared statements did not honor `zeroDateTimeBehavior` property, and would cause class-cast exceptions when using `ResultSet.getObject()`, as the all-zero string was always returned. (Bug#5235)
- Fixed batched updates with server prepared statements weren't looking if the types had changed for a given batched set of parameters compared to the previous set, causing the server to return the error "Wrong arguments to mysql_stmt_execute()".
- Handle case when string representation of timestamp contains trailing '.' with no numbers following it.
- Inefficient detection of pre-existing string instances in `ResultSet.getNativeString()`. (Bug#5706)
- Don't throw exceptions for `Connection.releaseSavepoint()`.
- Use a per-session `Calendar` instance by default when decoding dates from `ServerPreparedStatement` (set to old, less performant behavior by setting property `dynamicCalendars=true`).
- Added experimental configuration property `dontUnpackBinaryResults`, which delays unpacking binary result set values until they're asked for, and only creates object instances for non-numerical values (it is set to `false` by default). For some usecase/jvm combinations, this is friendlier on the garbage collector.
- `UNSIGNED BIGINT` unpacked incorrectly from server-side prepared statement result sets. (Bug#5729)
- `ServerSidePreparedStatement` allocating short-lived objects unnecessarily. (Bug#6225)
- Removed unwanted new `Throwable()` in `ResultSet` constructor due to bad merge (caused a new object instance that was never used for every result set created). Found while profiling for Bug#6359.
- Fixed too-early creation of `StringBuffer` in `EscapeProcessor.escapeSQL()`, also return `String` when escaping not needed (to avoid unnecessary object allocations). Found while profiling for Bug#6359.
- Use null-safe-equals for key comparisons in updatable result sets.
- `SUM()` on `DECIMAL` with server-side prepared statement ignores scale if zero-padding is needed (this ends up being due to conversion to `DOUBLE` by server, which when converted to a string to parse into `BigDecimal`, loses all "padding" zeros). (Bug#6537)
- Use `DatabaseMetaData.getIdentiferQuoteString()` when building DBMD queries.
- Use 1MB packet for sending file for `LOAD DATA LOCAL INFILE` if that is `< max_allowed_packet` on server.
- `ResultSetMetaData.getColumnDisplaySize()` returns incorrect values for multi-byte charsets. (Bug#6399)
- Make auto-deserialization of `java.lang.Objects` stored in `BLOB` columns configurable via `autoDeserialize` property (defaults to `false`).
- Re-work `Field.isOpaqueBinary()` to detect `CHAR(n) CHARACTER SET BINARY` to support fixed-length binary fields for `ResultSet.getObject()`.
- Use our own implementation of buffered input streams to get around blocking behavior of `java.io.BufferedInputStream`. Disable this with `useReadAheadInput=false`.
- Failing to connect to the server when one of the addresses for the given host name is IPV6 (which the server does not yet bind on). The driver now loops through *all* IP addresses for a given host, and stops on the first one that `accepts()` a `socket.connect()`. (Bug#6348)

C.5.3.12. Changes in MySQL Connector/J 3.1.4-beta (04 September 2004)

- Connector/J 3.1.3 beta does not handle integers correctly (caused by changes to support unsigned reads in `Buffer.readInt()` - `> Buffer.readShort()`). (Bug#4510)
- Added support in `DatabaseMetaData.getTables()` and `getTableTypes()` for views, which are now available in MySQL server 5.0.x.
- `ServerPreparedStatement.execute*()` sometimes threw `ArrayIndexOutOfBoundsException` when unpacking

field metadata. ([Bug#4642](#))

- Optimized integer number parsing, enable “old” slower integer parsing using JDK classes via `useFastIntParsing=false` property.
- Added `useOnlyServerErrorMessages` property, which causes message text in exceptions generated by the server to only contain the text sent by the server (as opposed to the `SQLState`’s “standard” description, followed by the server’s error message). This property is set to `true` by default.
- `ResultSet.wasNull()` does not work for primitives if a previous `null` was returned. ([Bug#4689](#))
- Track packet sequence numbers if `enablePacketDebug=true`, and throw an exception if packets received out-of-order.
- `ResultSet.getObject()` returns wrong type for strings when using prepared statements. ([Bug#4482](#))
- Calling `MysqlPooledConnection.close()` twice (even though an application error), caused NPE. Fixed.
- `ServerPreparedStatement` dealing with return of `DECIMAL` type don’t work. ([Bug#5012](#))
- `ResultSet.getObject()` doesn’t return type `Boolean` for pseudo-bit types from prepared statements on 4.1.x (shortcut for avoiding extra type conversion when using binary-encoded result sets obscured test in `getObject()` for “pseudo” bit type). ([Bug#5032](#))
- You can now use URLs in `LOAD DATA LOCAL INFILE` statements, and the driver will use Java’s built-in handlers for retrieving the data and sending it to the server. This feature is not enabled by default, you must set the `allowUrlInLocalInfile` connection property to `true`.
- The driver is more strict about truncation of numerics on `ResultSet.get*()`, and will throw an `SQLException` when truncation is detected. You can disable this by setting `jdbcCompliantTruncation` to `false` (it is enabled by default, as this functionality is required for JDBC compliance).
- Added three ways to deal with all-zero datetimes when reading them from a `ResultSet`: `exception` (the default), which throws an `SQLException` with an `SQLState` of `S1009`; `convertToNull`, which returns `NULL` instead of the date; and `round`, which rounds the date to the nearest closest value which is `'0001-01-01'`.
- Fixed `ServerPreparedStatement` to read prepared statement metadata off the wire, even though it’s currently a placeholder instead of using `MysqlIO.clearInputStream()` which didn’t work at various times because data wasn’t available to read from the server yet. This fixes sporadic errors users were having with `ServerPreparedStatement` throwing `ArrayIndexOutOfBoundsException`.
- Use `com.mysql.jdbc.Message`’s classloader when loading resource bundle, should fix sporadic issues when the caller’s classloader can’t locate the resource bundle.

C.5.3.13. Changes in MySQL Connector/J 3.1.3-beta (07 July 2004)

- Mangle output parameter names for `CallableStatements` so they will not clash with user variable names.
- Added support for `INOUT` parameters in `CallableStatements`.
- Null bitmask sent for server-side prepared statements was incorrect. ([Bug#4119](#))
- Use SQL Standard SQL states by default, unless `useSqlStateCodes` property is set to `false`.
- Added packet debugging code (see the `enablePacketDebug` property documentation).
- Added constants for MySQL error numbers (publicly accessible, see `com.mysql.jdbc.MysqlErrorNumbers`), and the ability to generate the mappings of vendor error codes to `SQLStates` that the driver uses (for documentation purposes).
- Externalized more messages (on-going effort).
- Error in retrieval of `mediumint` column with prepared statements and binary protocol. ([Bug#4311](#))
- Support new time zone variables in MySQL-4.1.3 when `useTimezone=true`.

- Support for unsigned numerics as return types from prepared statements. This also causes a change in `ResultSet.getObject()` for the `bigint unsigned` type, which used to return `BigDecimal` instances, it now returns instances of `java.lang.BigInteger`.

C.5.3.14. Changes in MySQL Connector/J 3.1.2-alpha (09 June 2004)

- Fixed stored procedure parameter parsing info when size was specified for a parameter (for example, `char()`, `varchar()`).
- Enabled callable statement caching via `cacheCallableStmts` property.
- Fixed case when no output parameters specified for a stored procedure caused a bogus query to be issued to retrieve out parameters, leading to a syntax error from the server.
- Fixed case when no parameters could cause a `NullPointerException` in `CallableStatement.setOutputParameters()`.
- Removed wrapping of exceptions in `MysqlIO.changeUser()`.
- Fixed sending of split packets for large queries, enabled nio ability to send large packets as well.
- Added `.toString()` functionality to `ServerPreparedStatement`, which should help if you're trying to debug a query that is a prepared statement (it shows SQL as the server would process).
- Added `gatherPerformanceMetrics` property, along with properties to control when/where this info gets logged (see docs for more info).
- `ServerPreparedStatements` weren't actually de-allocating server-side resources when `.close()` was called.
- Added `logSlowQueries` property, along with `slowQueriesThresholdMillis` property to control when a query should be considered "slow."
- Correctly map output parameters to position given in `prepareCall()` versus. order implied during `registerOutParameter()`. (Bug#3146)
- Correctly detect initial character set for servers $\geq 4.1.0$.
- Cleaned up detection of server properties.
- Support placeholder for parameter metadata for server $\geq 4.1.2$.
- `getProcedures()` does not return any procedures in result set. (Bug#3539)
- `getProcedureColumns()` doesn't work with wildcards for procedure name. (Bug#3540)
- `DBMD.getSQLStateType()` returns incorrect value. (Bug#3520)
- Added `connectionCollation` property to cause driver to issue `set collation_connection=...` query on connection init if default collation for given charset is not appropriate.
- Fixed `DatabaseMetaData.getProcedures()` when run on MySQL-5.0.0 (output of `SHOW PROCEDURE STATUS` changed between 5.0.0 and 5.0.1).
- `getWarnings()` returns `SQLWarning` instead of `DataTruncation`. (Bug#3804)
- Don't enable server-side prepared statements for server version 5.0.0 or 5.0.1, as they aren't compatible with the '4.1.2+' style that the driver uses (the driver expects information to come back that isn't there, so it hangs).

C.5.3.15. Changes in MySQL Connector/J 3.1.1-alpha (14 February 2004)

- Fixed bug with `UpdatableResultSets` not using client-side prepared statements.
- Fixed character encoding issues when converting bytes to ASCII when MySQL doesn't provide the character set, and the JVM is set

to a multi-byte encoding (usually affecting retrieval of numeric values).

- Unpack “unknown” data types from server prepared statements as `Strings`.
- Implemented long data (Blobs, Clobs, InputStreams, Readers) for server prepared statements.
- Implemented `Statement.getWarnings()` for MySQL-4.1 and newer (using `SHOW WARNINGS`).
- Default result set type changed to `TYPE_FORWARD_ONLY` (JDBC compliance).
- Centralized setting of result set type and concurrency.
- Refactored how connection properties are set and exposed as `DriverPropertyInfo` as well as `Connection` and `Data-Source` properties.
- Support for NIO. Use `useNIO=true` on platforms that support NIO.
- Support for transaction savepoints (MySQL >= 4.0.14 or 4.1.1).
- Support for `mysql_change_user()`. See the `changeUser()` method in `com.mysql.jdbc.Connection`.
- Reduced number of methods called in average query to be more efficient.
- Prepared `Statements` will be re-prepared on auto-reconnect. Any errors encountered are postponed until first attempt to re-execute the re-prepared statement.
- Ensure that warnings are cleared before executing queries on prepared statements, as-per JDBC spec (now that we support warnings).
- Support “old” `profileSql` capitalization in `ConnectionProperties`. This property is deprecated, you should use `profileSQL` if possible.
- Optimized `Buffer.readLenByteArray()` to return shared empty byte array when length is 0.
- Allow contents of `PreparedStatement.setBlob()` to be retained between calls to `.execute*()`.
- Deal with 0-length tokens in `EscapeProcessor` (caused by callable statement escape syntax).
- Check for closed connection on delete/update/insert row operations in `UpdatableResultSet`.
- Fix support for table aliases when checking for all primary keys in `UpdatableResultSet`.
- Removed `useFastDates` connection property.
- Correctly initialize datasource properties from JNDI Refs, including explicitly specified URLs.
- `DatabaseMetaData` now reports `supportsStoredProcedures()` for MySQL versions >= 5.0.0
- Fixed stack overflow in `Connection.prepareCall()` (bad merge).
- Fixed `IllegalAccessError` to `Calendar.getTimeInMillis()` in `DateTimeValue` (for JDK < 1.4).
- `DatabaseMetaData.getColumns()` is not returning correct column ordinal info for non-`'%'` column name patterns. ([Bug#1673](#))
- Merged fix of datatype mapping from MySQL type `FLOAT` to `java.sql.Types.REAL` from 3.0 branch.
- Detect collation of column for `RSMD.isCaseSensitive()`.
- Fixed sending of queries larger than 16M.
- Added named and indexed input/output parameter support to `CallableStatement`. MySQL-5.0.x or newer.
- Fixed `NullPointerException` in `ServerPreparedStatement.setTimestamp()`, as well as year and month discrepancies in `ServerPreparedStatement.setTimestamp()`, `setDate()`.
- Added ability to have multiple database/JVM targets for compliance and regression/unit tests in `build.xml`.

- Fixed NPE and year/month bad conversions when accessing some datetime functionality in `ServerPreparedStatement` and their resultant result sets.
- Display where/why a connection was implicitly closed (to aid debugging).
- `CommunicationsException` implemented, that tries to determine why communications was lost with a server, and displays possible reasons when `.getMessage()` is called.
- `NULL` values for numeric types in binary encoded result sets causing `NullPointerExceptions`. ([Bug#2359](#))
- Implemented `Connection.prepareCall()`, and `DatabaseMetaData.getProcedures()` and `getProcedureColumns()`.
- Reset `long binary` parameters in `ServerPreparedStatement` when `clearParameters()` is called, by sending `COM_RESET_STMT` to the server.
- Merged prepared statement caching, and `.getMetaData()` support from 3.0 branch.
- Fixed off-by-1900 error in some cases for years in `TimeUtil.fastDate/TimeCreate()` when unpacking results from server-side prepared statements.
- Fixed charset conversion issue in `getTables()`. ([Bug#2502](#))
- Implemented multiple result sets returned from a statement or stored procedure.
- Server-side prepared statements were not returning datatype `YEAR` correctly. ([Bug#2606](#))
- Enabled streaming of result sets from server-side prepared statements.
- Class-cast exception when using scrolling result sets and server-side prepared statements. ([Bug#2623](#))
- Merged unbuffered input code from 3.0.
- Fixed `ConnectionProperties` that weren't properly exposed via accessors, cleaned up `ConnectionProperties` code.
- `NULL` fields were not being encoded correctly in all cases in server-side prepared statements. ([Bug#2671](#))
- Fixed rare buffer underflow when writing numbers into buffers for sending prepared statement execution requests.
- Use DocBook version of docs for shipped versions of drivers.

C.5.3.16. Changes in MySQL Connector/J 3.1.0-alpha (18 February 2003)

- Added `requireSSL` property.
- Added `useServerPrepStmts` property (default `false`). The driver will use server-side prepared statements when the server version supports them (4.1 and newer) when this property is set to `true`. It is currently set to `false` by default until all bind/fetch functionality has been implemented. Currently only DML prepared statements are implemented for 4.1 server-side prepared statements.
- Track open `Statements`, close all when `Connection.close()` is called (JDBC compliance).

C.5.4. Changes in MySQL Connector/J 3.0.x

C.5.4.1. Changes in MySQL Connector/J 3.0.17-ga (23 June 2005)

- `Timestamp/Time` conversion goes in the wrong “direction” when `useTimeZone=true` and server time zone differs from client time zone. ([Bug#5874](#))
- `DatabaseMetaData.getIndexInfo()` ignored `unique` parameter. ([Bug#7081](#))

- Support new protocol type `MYSQL_TYPE_VARCHAR`.
- Added `useOldUTF8Behavior` configuration property, which causes JDBC driver to act like it did with MySQL-4.0.x and earlier when the character encoding is `utf-8` when connected to MySQL-4.1 or newer.
- Statements created from a pooled connection were returning physical connection instead of logical connection when `getConnection()` was called. (Bug#7316)
- `PreparedStatement` don't encode Big5 (and other multi-byte) character sets correctly in static SQL strings. (Bug#7033)
- Connections starting up failed-over (due to down master) never retry master. (Bug#6966)
- `PreparedStatement.fixDecimalExponent()` adding extra `+`, making number unparseable by MySQL server. (Bug#7061)
- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. (Bug#7686)
- Backported `SQLState` codes mapping from Connector/J 3.1, enable with `useSqlStateCodes=true` as a connection property, it defaults to `false` in this release, so that we don't break legacy applications (it defaults to `true` starting with Connector/J 3.1).
- `PreparedStatement.fixDecimalExponent()` adding extra `+`, making number unparseable by MySQL server. (Bug#7601)
- Escape sequence `{fn convert(..., type)}` now supports ODBC-style types that are prepended by `SQL_`.
- Fixed duplicated code in `configureClientCharset()` that prevented `useOldUTF8Behavior=true` from working properly.
- Handle streaming result sets with more than 2 billion rows properly by fixing wraparound of row number counter.
- `MS932`, `SHIFT_JIS`, and `Windows_31J` not recognized as aliases for `sjis`. (Bug#7607)
- Adding `CP943` to aliases for `sjis`. (Bug#6549, fixed while fixing Bug#7607)
- Which requires hex escaping of binary data when using multi-byte charsets with prepared statements. (Bug#8064)
- `NON_UNIQUE` column from `DBMD.getIndexInfo()` returned inverted value. (Bug#8812)
- Workaround for server Bug#9098: Default values of `CURRENT_*` for `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` columns can't be distinguished from `string` values, so `UpdatableResultSet.moveToInsertRow()` generates bad SQL for inserting default values.
- `EUCKR` charset is sent as `SET NAMES euc_kr` which MySQL-4.1 and newer doesn't understand. (Bug#8629)
- `DatabaseMetaData.supportsSelectForUpdate()` returns correct value based on server version.
- Use hex escapes for `PreparedStatement.setBytes()` for double-byte charsets including “aliases” `Windows-31J`, `CP934`, `MS932`.
- Added support for the `EUC_JP_Solaris` character encoding, which maps to a MySQL encoding of `eucjpms` (backported from 3.1 branch). This only works on servers that support `eucjpms`, namely 5.0.3 or later.

C.5.4.2. Changes in MySQL Connector/J 3.0.16-ga (15 November 2004)

- Re-issue character set configuration commands when re-using pooled connections and/or `Connection.changeUser()` when connected to MySQL-4.1 or newer.
- Fixed `ResultSetMetaData.isReadOnly()` to detect non-writable columns when connected to MySQL-4.1 or newer, based on existence of “original” table and column names.
- `ResultSet.updateByte()` when on insert row throws `ArrayOutOfBoundsException`. (Bug#5664)
- Fixed `DatabaseMetaData.getTypes()` returning incorrect (this is, non-negative) scale for the `NUMERIC` type.

- Off-by-one bug in `Buffer.readString(string)`. (Bug#5664)
- Made `TINYINT(1)` -> `BIT/Boolean` conversion configurable via `tinyInt1isBit` property (default `true` to be JDBC compliant out of the box).
- Only set `character_set_results` during connection establishment if server version $\geq 4.1.1$.
- Fixed regression where `useUnbufferedInput` was defaulting to `false`.
- `ResultSet.getTimestamp()` on a column with `TIME` in it fails. (Bug#5664)

C.5.4.3. Changes in MySQL Connector/J 3.0.15-production (04 September 2004)

- `StringUtils.escapeEasternUnicodeByteStream` was still broken for GBK. (Bug#4010)
- Failover for `autoReconnect` not using port numbers for any hosts, and not retrying all hosts. (**Warning:** This required a change to the `SocketFactory.connect()` method signature, which is now `public Socket connect(String host, int portNumber, Properties props)`; therefore, any third-party socket factories will have to be changed to support this signature. (Bug#4334)
- Logical connections created by `MysqlConnectionPoolDataSource` will now issue a `rollback()` when they are closed and sent back to the pool. If your application server/connection pool already does this for you, you can set the `rollbackOnPooledClose` property to `false` to avoid the overhead of an extra `rollback()`.
- Removed redundant calls to `checkRowPos()` in `ResultSet`.
- `DOUBLE` mapped twice in `DBMD.getTypeInfo()`. (Bug#4742)
- Added FLOSS license exemption.
- Calling `.close()` twice on a `PooledConnection` causes NPE. (Bug#4808)
- `DBMD.getColumns()` returns incorrect JDBC type for unsigned columns. This affects type mappings for all numeric types in the `RSMD.getColumnType()` and `RSMD.getColumnTypeNames()` methods as well, to ensure that “like” types from `DBMD.getColumns()` match up with what `RSMD.getColumnType()` and `getColumnTypeNames()` return. (Bug#4138, Bug#4860)
- “Production” is now “GA” (General Availability) in naming scheme of distributions.
- `RSMD.getPrecision()` returning 0 for non-numeric types (should return max length in chars for non-binary types, max length in bytes for binary types). This fix also fixes mapping of `RSMD.getColumnType()` and `RSMD.getColumnTypeName()` for the `BLOB` types based on the length sent from the server (the server doesn't distinguish between `TINYBLOB`, `BLOB`, `MEDIUMBLOB` or `LONGBLOB` at the network protocol level). (Bug#4880)
- `ResultSet` should release `Field[]` instance in `.close()`. (Bug#5022)
- `ResultSet.getMetaData()` should not return incorrectly initialized metadata if the result set has been closed, but should instead throw an `SQLException`. Also fixed for `getRow()` and `getWarnings()` and traversal methods by calling `checkClosed()` before operating on instance-level fields that are nullified during `.close()`. (Bug#5069)
- Parse new time zone variables from 4.1.x servers.
- Use `_binary` introducer for `PreparedStatement.setBytes()` and `set*Stream()` when connected to MySQL-4.1.x or newer to avoid misinterpretation during character conversion.

C.5.4.4. Changes in MySQL Connector/J 3.0.14-production (28 May 2004)

- Fixed URL parsing error.

C.5.4.5. Changes in MySQL Connector/J 3.0.13-production (27 May 2004)

- Using a `MySQLDataSource` without server name fails. (Bug#3848)
- `No Database Selected` when using `MysqlConnectionPoolDataSource`. (Bug#3920)
- `PreparedStatement.getGeneratedKeys()` method returns only 1 result for batched insertions. (Bug#3873)

C.5.4.6. Changes in MySQL Connector/J 3.0.12-production (18 May 2004)

- Add unsigned attribute to `DatabaseMetaData.getColumns()` output in the `TYPE_NAME` column.
- Added `failOverReadOnly` property, to allow end-user to configure state of connection (read-only/writable) when failed over.
- Backported “change user” and “reset server state” functionality from 3.1 branch, to allow clients of `MysqlConnectionPoolDataSource` to reset server state on `getConnection()` on a pooled connection.
- Don't escape SJIS/GBK/BIG5 when using MySQL-4.1 or newer.
- Allow `url` parameter for `MysqlDataSource` and `MysqlConnectionPool DataSource` so that passing of other properties is possible from inside appservers.
- Map duplicate key and foreign key errors to SQLState of `23000`.
- Backport documentation tooling from 3.1 branch.
- Return creating statement for `ResultSets` created by `getGeneratedKeys()`. (Bug#2957)
- Allow `java.util.Date` to be sent in as parameter to `PreparedStatement.setObject()`, converting it to a `Timestamp` to maintain full precision. (Bug#103).
- Don't truncate `BLOB` or `CLOB` values when using `setBytes()` and/or `setBinary/CharacterStream()`. (Bug#2670).
- Dynamically configure character set mappings for field-level character sets on MySQL-4.1.0 and newer using `SHOW COLLATION` when connecting.
- Map `binary` character set to `US-ASCII` to support `DATETIME` charset recognition for servers $\geq 4.1.2$.
- Use `SET character_set_results` during initialization to allow any charset to be returned to the driver for result sets.
- Use `charsetnr` returned during connect to encode queries before issuing `SET NAMES` on MySQL $\geq 4.1.0$.
- Add helper methods to `ResultSetMetaData` (`getColumnCharacterEncoding()` and `getColumnCharacterSet()`) to allow end-users to see what charset the driver thinks it should be using for the column.
- Only set `character_set_results` for MySQL $\geq 4.1.0$.
- `StringUtils.escapeSJISByteStream()` not covering all eastern double-byte charsets correctly. (Bug#3511)
- Renamed `StringUtils.escapeSJISByteStream()` to more appropriate `escapeEasternUnicodeByteStream()`.
- Not specifying database in URL caused `MalformedURLException` exception. (Bug#3554)
- Auto-convert MySQL encoding names to Java encoding names if used for `characterEncoding` property.
- Added encoding names that are recognized on some JVMs to fix case where they were reverse-mapped to MySQL encoding names incorrectly.
- Use `junit.textui.TestRunner` for all unit tests (to allow them to be run from the command line outside of Ant or Eclipse).
- `UpdatableResultSet` not picking up default values for `moveToInsertRow()`. (Bug#3557)
- Inconsistent reporting of data type. The server still doesn't return all types for `*BLOBs` `*TEXT` correctly, so the driver won't return those correctly. (Bug#3570)
- `DBMD.getSQLStateType()` returns incorrect value. (Bug#3520)

- Fixed regression in `PreparedStatement.setString()` and eastern character encodings.
- Made `StringRegressionTest` 4.1-unicode aware.

C.5.4.7. Changes in MySQL Connector/J 3.0.11-stable (19 February 2004)

- Trigger a `SET NAMES utf8` when encoding is forced to `utf8` or `utf-8` via the `characterEncoding` property. Previously, only the Java-style encoding name of `utf-8` would trigger this.
- `AutoReconnect` time was growing faster than exponentially. (Bug#2447)
- Fixed failover always going to last host in list. (Bug#2578)
- Added `useUnbufferedInput` parameter, and now use it by default (due to JVM issue <http://developer.java.sun.com/developer/bugParade/bugs/4401235.html>)
- Detect `on/off` or `1, 2, 3` form of `lower_case_table_names` value on server.
- Return `java.lang.Integer` for `TINYINT` and `SMALLINT` types from `ResultSetMetaData.getColumnClassName()`. (Bug#2852)
- Return `java.lang.Double` for `FLOAT` type from `ResultSetMetaData.getColumnClassName()`. (Bug#2855)
- Return `[B` instead of `java.lang.Object` for `BINARY`, `VARBINARY` and `LONGVARBINARY` types from `ResultSetMetaData.getColumnClassName()` (JDBC compliance).
- Issue connection events on all instances created from a `ConnectionPoolDataSource`.

C.5.4.8. Changes in MySQL Connector/J 3.0.10-stable (13 January 2004)

- Don't count quoted IDs when inside a 'string' in `PreparedStatement` parsing. (Bug#1511)
- “Friendlier” exception message for `PacketTooLargeException`. (Bug#1534)
- Backported fix for aliased tables and `UpdatableResultSets` in `checkUpdatability()` method from 3.1 branch.
- Fix for `ArrayIndexOutOfBoundsException` exception when using `Statement.setMaxRows()`. (Bug#1695)
- Barge blobs and split packets not being read correctly. (Bug#1576)
- Fixed regression of `Statement.getGeneratedKeys()` and `REPLACE` statements.
- Subsequent call to `ResultSet.updateFoo()` causes NPE if result set is not updatable. (Bug#1630)
- Fix for 4.1.1-style authentication with no password.
- Foreign Keys column sequence is not consistent in `DatabaseMetaData.getImported/Exported/CrossReference()`. (Bug#1731)
- `DatabaseMetaData.getSystemFunction()` returning bad function `VResultsSion`. (Bug#1775)
- Cross-database updatable result sets are not checked for updatability correctly. (Bug#1592)
- `DatabaseMetaData.getColumns()` should return `Types.LONGVARCHAR` for MySQL `LONGTEXT` type.
- `ResultSet.getObject()` on `TINYINT` and `SMALLINT` columns should return Java type `Integer`. (Bug#1913)
- Added `alwaysClearStream` connection property, which causes the driver to always empty any remaining data on the input stream before each query.
- Added more descriptive error message `Server Configuration Denies Access to DataSource`, as well as retrieval of message from server.

- Autoreconnect code didn't set catalog upon reconnect if it had been changed.
- Implement `ResultSet.updateClob()`.
- `ResultSetMetaData.isCaseSensitive()` returned wrong value for `CHAR/VARCHAR` columns.
- Connection property `maxRows` not honored. (Bug#1933)
- Statements being created too many times in `DBMD.extractForeignKeyFromCreateTable()`. (Bug#1925)
- Support escape sequence `{fn convert ... }`. (Bug#1914)
- `ArrayIndexOutOfBoundsException` when parameter number == number of parameters + 1. (Bug#1958)
- `ResultSet.findColumn()` should use first matching column name when there are duplicate column names in `SELECT` query (JDBC-compliance). (Bug#2006)
- Removed static synchronization bottleneck from `PreparedStatement.setTimestamp()`.
- Removed static synchronization bottleneck from instance factory method of `SingleByteCharsetConverter`.
- Enable caching of the parsing stage of prepared statements via the `cachePrepStmts`, `prepStmtCacheSize`, and `prepStmtCacheSqlLimit` properties (disabled by default).
- Speed up parsing of `PreparedStatements`, try to use one-pass whenever possible.
- Fixed security exception when used in Applets (applets can't read the system property `file.encoding` which is needed for `LOAD DATA LOCAL INFILE`).
- Use constants for `SQLStates`.
- Map charset `ko18_ru` to `ko18r` when connected to MySQL-4.1.0 or newer.
- Ensure that `Buffer.writeString()` saves room for the `\0`.
- Fixed exception `Unknown character set 'danish'` on connect with JDK-1.4.0
- Fixed mappings in `SQLError` to report deadlocks with `SQLStates` of `41000`.
- `maxRows` property would affect internal statements, so check it for all statement creation internal to the driver, and set to 0 when it is not.

C.5.4.9. Changes in MySQL Connector/J 3.0.9-stable (07 October 2003)

- Faster date handling code in `ResultSet` and `PreparedStatement` (no longer uses `Date` methods that synchronize on static calendars).
- Fixed test for end of buffer in `Buffer.readString()`.
- Fixed `ResultSet.previous()` behavior to move current position to before result set when on first row of result set. (Bug#496)
- Fixed `Statement` and `PreparedStatement` issuing bogus queries when `setMaxRows()` had been used and a `LIMIT` clause was present in the query.
- `refreshRow` didn't work when primary key values contained values that needed to be escaped (they ended up being doubly escaped). (Bug#661)
- Support `InnoDB` constraint names when extracting foreign key information in `DatabaseMetaData` (implementing ideas from Parwinder Sekhon). (Bug#517, Bug#664)
- Backported 4.1 protocol changes from 3.1 branch (server-side `SQL` states, new field information, larger client capability flags, connect-with-database, and so forth).
- Fix `UpdatableResultSet` to return values for `getXXX()` when on insert row. (Bug#675)

- The `insertRow` in an `UpdatableResultSet` is now loaded with the default column values when `moveToInsertRow()` is called. (Bug#688)
- `DatabaseMetaData.getColumns()` wasn't returning `NULL` for default values that are specified as `NULL`.
- Change default statement type/concurrency to `TYPE_FORWARD_ONLY` and `CONCUR_READ_ONLY` (spec compliance).
- Don't try and reset isolation level on reconnect if MySQL doesn't support them.
- Don't wrap `SQLExceptions` in `RowDataDynamic`.
- Don't change timestamp TZ twice if `useTimezone==true`. (Bug#774)
- Fixed regression in large split-packet handling. (Bug#848)
- Better diagnostic error messages in exceptions for “streaming” result sets.
- Issue exception on `ResultSet.getXXX()` on empty result set (wasn't caught in some cases).
- Don't hide messages from exceptions thrown in I/O layers.
- Don't fire connection closed events when closing pooled connections, or on `PooledConnection.getConnection()` with already open connections. (Bug#884)
- Clip +/- INF (to smallest and largest representative values for the type in MySQL) and NaN (to 0) for `setDouble/setFloat()`, and issue a warning on the statement when the server does not support +/- INF or NaN.
- Double-escaping of `'\'` when charset is SJIS or GBK and `'\'` appears in non-escaped input. (Bug#879)
- When emptying input stream of unused rows for “streaming” result sets, have the current thread `yield()` every 100 rows in order to not monopolize CPU time.
- `DatabaseMetaData.getColumns()` getting confused about the keyword “set” in character columns. (Bug#1099)
- Fixed deadlock issue with `Statement.setMaxRows()`.
- Fixed `CLOB.truncate()`. (Bug#1130)
- Optimized `CLOB.setCharacterStream()`. (Bug#1131)
- Made `databaseName`, `portNumber`, and `serverName` optional parameters for `MySQLDataSourceFactory`. (Bug#1246)
- `ResultSet.get/setString` mashing char 127. (Bug#1247)
- Backported authentication changes for 4.1.1 and newer from 3.1 branch.
- Added `com.mysql.jdbc.util.BaseBugReport` to help creation of testcases for bug reports.
- Added property to “clobber” streaming results, by setting the `clobberStreamingResults` property to `true` (the default is `false`). This will cause a “streaming” `ResultSet` to be automatically closed, and any outstanding data still streaming from the server to be discarded if another query is executed before all the data has been read from the server.

C.5.4.10. Changes in MySQL Connector/J 3.0.8-stable (23 May 2003)

- Allow bogus URLs in `Driver.getPropertyInfo()`.
- Return list of generated keys when using multi-value `INSERTS` with `Statement.getGeneratedKeys()`.
- Use JVM charset with filenames and `LOAD DATA [LOCAL] INFILE`.
- Fix infinite loop with `Connection.cleanup()`.
- Changed Ant target `compile-core` to `compile-driver`, and made testsuite compilation a separate target.

- Fixed result set not getting set for `Statement.executeUpdate()`, which affected `getGeneratedKeys()` and `getUpdateCount()` in some cases.
- Unicode character 0xFFFF in a string would cause the driver to throw an `ArrayOutOfBoundsException`. ([Bug#378](#)).
- Return correct number of generated keys when using `REPLACE` statements.
- Fix problem detecting server character set in some cases.
- Fix row data decoding error when using very large packets.
- Optimized row data decoding.
- Issue exception when operating on an already closed prepared statement.
- Fixed SJIS encoding bug, thanks to Naoto Sato.
- Optimized usage of `EscapeProcessor`.
- Allow multiple calls to `Statement.close()`.

C.5.4.11. Changes in MySQL Connector/J 3.0.7-stable (08 April 2003)

- Fixed `MysqlPooledConnection.close()` calling wrong event type.
- Fixed `StringIndexOutOfBoundsException` in `PreparedStatement.setClob()`.
- 4.1 Column Metadata fixes.
- Remove synchronization from `Driver.connect()` and `Driver.acceptsUrl()`.
- `IOExceptions` during a transaction now cause the `Connection` to be closed.
- Fixed missing conversion for `YEAR` type in `ResultSetMetaData.getColumnTypeName()`.
- Don't pick up indexes that start with `pri` as primary keys for `DBMD.getPrimaryKeys()`.
- Throw `SQLExceptions` when trying to do operations on a forcefully closed `Connection` (that is, when a communication link failure occurs).
- You can now toggle profiling on/off using `Connection.setProfileSql(boolean)`.
- Fixed charset issues with database metadata (charset was not getting set correctly).
- Updatable `ResultSets` can now be created for aliased tables/columns when connected to MySQL-4.1 or newer.
- Fixed `LOAD DATA LOCAL INFILE` bug when file > `max_allowed_packet`.
- Fixed escaping of 0x5c ('\\') character for GBK and Big5 charsets.
- Fixed `ResultSet.getTimestamp()` when underlying field is of type `DATE`.
- Ensure that packet size from `alignPacketSize()` does not exceed `max_allowed_packet` (JVM bug)
- Don't reset `Connection.isReadOnly()` when autoReconnecting.

C.5.4.12. Changes in MySQL Connector/J 3.0.6-stable (18 February 2003)

- Fixed `ResultSetMetaData` to return "" when catalog not known. Fixes `NullPointerExceptions` with Sun's `CachedRowSet`.
- Fixed `DBMD.getTypeInfo()` and `DBMD.getColumns()` returning different value for precision in `TEXT` and `BLOB` types.

- Allow ignoring of warning for “non transactional tables” during rollback (compliance/usability) by setting `ignoreNonTxTables` property to `true`.
- Fixed `SQLExceptions` getting swallowed on initial connect.
- Fixed `Statement.setMaxRows()` to stop sending `LIMIT` type queries when not needed (performance).
- Clean up `Statement` query/method mismatch tests (that is, `INSERT` not allowed with `.executeQuery()`).
- More checks added in `ResultSet` traversal method to catch when in closed state.
- Fixed `ResultSetMetaData.isWritable()` to return correct value.
- Add “window” of different `NULL` sorting behavior to `DBMD.nullsAreSortedAtStart` (4.0.2 to 4.0.10, true; otherwise, no).
- Implemented `Blob.setBytes()`. You still need to pass the resultant `Blob` back into an updatable `ResultSet` or `PreparedStatement` to persist the changes, because MySQL does not support “locators”.
- Backported 4.1 charset field info changes from Connector/J 3.1.

C.5.4.13. Changes in MySQL Connector/J 3.0.5-gamma (22 January 2003)

- Fixed `Buffer.fastSkipLenString()` causing `ArrayIndexOutOfBoundsException` exceptions with some queries when unpacking fields.
- Implemented an empty `TypeMap` for `Connection.getTypeMap()` so that some third-party apps work with MySQL (IBM WebSphere 5.0 Connection pool).
- Added missing `LONGTEXT` type to `DBMD.getColumns()`.
- Retrieve `TX_ISOLATION` from database for `Connection.getTransactionIsolation()` when the MySQL version supports it, instead of an instance variable.
- Quote table names in `DatabaseMetaData.getColumns()`, `getPrimaryKeys()`, `getIndexInfo()`, `getBestRowIdentifier()`.
- Greatly reduce memory required for `setBinaryStream()` in `PreparedStatement`s.
- Fixed `ResultSet.isBeforeFirst()` for empty result sets.
- Added update options for foreign key metadata.

C.5.4.14. Changes in MySQL Connector/J 3.0.4-gamma (06 January 2003)

- Added quoted identifiers to database names for `Connection.setCatalog`.
- Added support for quoted identifiers in `PreparedStatement` parser.
- Streamlined character conversion and `byte[]` handling in `PreparedStatement`s for `setByte()`.
- Reduce memory footprint of `PreparedStatement`s by sharing outbound packet with `MysqlIO`.
- Added `strictUpdates` property to allow control of amount of checking for “correctness” of updatable result sets. Set this to `false` if you want faster updatable result sets and you know that you create them from `SELECT` statements on tables with primary keys and that you have selected all primary keys in your query.
- Added support for 4.0.8-style large packets.
- Fixed `PreparedStatement.executeBatch()` parameter overwriting.

C.5.4.15. Changes in MySQL Connector/J 3.0.3-dev (17 December 2002)

- Changed `charsToByte` in `SingleByteCharConverter` to be non-static.
- Changed `SingleByteCharConverter` to use lazy initialization of each converter.
- Fixed charset handling in `Fields.java`.
- Implemented `Connection.nativeSQL()`.
- More robust escape tokenizer: Recognize `--` comments, and allow nested escape sequences (see `test-suite.EscapeProcessingTest`).
- `DBMD.getImported/ExportedKeys()` now handles multiple foreign keys per table.
- Fixed `ResultSetMetaData.getPrecision()` returning incorrect values for some floating-point types.
- Fixed `ResultSetMetaData.getColumnTypeName()` returning `BLOB` for `TEXT` and `TEXT` for `BLOB` types.
- Fixed `Buffer.isLastDataPacket()` for 4.1 and newer servers.
- Added `CLIENT_LONG_FLAG` to be able to get more column flags (`isAutoIncrement()` being the most important).
- Because of above, implemented `ResultSetMetaData.isAutoIncrement()` to use `Field.isAutoIncrement()`.
- Honor `lower_case_table_names` when enabled in the server when doing table name comparisons in `DatabaseMetaData` methods.
- Some MySQL-4.1 protocol support (extended field info from selects).
- Use non-aliased table/column names and database names to fully qualify tables and columns in `UpdatableResultSet` (requires MySQL-4.1 or newer).
- Allow user to alter behavior of `Statement/PreparedStatement.executeBatch()` via `continueBatchOnError` property (defaults to `true`).
- Check for connection closed in more `Connection` methods (`createStatement`, `prepareStatement`, `setTransactionIsolation`, `setAutoCommit`).
- More robust implementation of updatable result sets. Checks that *all* primary keys of the table have been selected.
- `LOAD DATA LOCAL INFILE ...` now works, if your server is configured to allow it. Can be turned off with the `allowLoadLocalInfile` property (see the `README`).
- Substitute `'?'` for unknown character conversions in single-byte character sets instead of `'\0'`.
- `NamedPipeSocketFactory` now works (only intended for Windows), see `README` for instructions.

C.5.4.16. Changes in MySQL Connector/J 3.0.2-dev (08 November 2002)

- Fixed issue with updatable result sets and `PreparedStatement`s not working.
- Fixed `ResultSet.setFetchDirection(FETCH_UNKNOWN)`.
- Fixed issue when calling `Statement.setFetchSize()` when using arbitrary values.
- Fixed incorrect conversion in `ResultSet.getLong()`.
- Implemented `ResultSet.updateBlob()`.
- Removed duplicate code from `UpdatableResultSet` (it can be inherited from `ResultSet`, the extra code for each method to handle updatability I thought might someday be necessary has not been needed).
- Fixed `UnsupportedEncodingException` thrown when “forcing” a character encoding via properties.
- Fixed various non-ASCII character encoding issues.

- Added driver property `useHostsInPrivileges`. Defaults to `true`. Affects whether or not `@hostname` will be used in `DBMD.getColumn/TablePrivileges`.
- All `DBMD` result set columns describing schemas now return `NULL` to be more compliant with the behavior of other JDBC drivers for other database systems (MySQL does not support schemas).
- Added SSL support. See [README](#) for information on how to use it.
- Properly restore connection properties when `autoReconnecting` or failing-over, including `autoCommit` state, and isolation level.
- Use `SHOW CREATE TABLE` when possible for determining foreign key information for `DatabaseMetaData`. Also allows cascade options for `DELETE` information to be returned.
- Escape `0x5c` character in strings for the SJIS charset.
- Fixed start position off-by-1 error in `Clob.getSubString()`.
- Implemented `Clob.truncate()`.
- Implemented `Clob.setString()`.
- Implemented `Clob.setAsciiStream()`.
- Implemented `Clob.setCharacterStream()`.
- Added `com.mysql.jdbc.MinorAdmin` class, which allows you to send `shutdown` command to MySQL server. This is intended to be used when “embedding” Java and MySQL server together in an end-user application.
- Added `connectTimeout` parameter that allows users of JDK-1.4 and newer to specify a maximum time to wait to establish a connection.
- Failover and `autoReconnect` work only when the connection is in an `autoCommit(false)` state, in order to stay transaction-safe.
- Added `queriesBeforeRetryMaster` property that specifies how many queries to issue when failed over before attempting to reconnect to the master (defaults to 50).
- Fixed `DBMD.supportsResultSetConcurrency()` so that it returns `true` for `ResultSet.TYPE_SCROLL_INSENSITIVE` and `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`.
- Fixed `ResultSet.isLast()` for empty result sets (should return `false`).
- `PreparedStatement` now honors stream lengths in `setBinary/Ascii/Character Stream()` unless you set the connection property `useStreamLengthsInPrepStmts` to `false`.
- Removed some not-needed temporary object creation by smarter use of `Strings` in `EscapeProcessor`, `Connection` and `DatabaseMetaData` classes.

C.5.4.17. Changes in MySQL Connector/J 3.0.1-dev (21 September 2002)

- Fixed `ResultSet.getRow()` off-by-one bug.
- Fixed `RowDataStatic.getAt()` off-by-one bug.
- Added limited `Clob` functionality (`ResultSet.getClob()`, `PreparedStatement.setClob()`, `PreparedStatement.setObject(Clob)`).
- Added `socketTimeout` parameter to URL.
- `Connection.isClosed()` no longer “pings” the server.
- `Connection.close()` issues `rollback()` when `getAutoCommit()` is `false`.
- Added `paranoid` parameter, which sanitizes error messages by removing “sensitive” information from them (such as hostnames,

ports, or usernames), as well as clearing “sensitive” data structures when possible.

- Fixed `ResultSetMetaData.isSigned()` for `TINYINT` and `BIGINT`.
- Charsets now automatically detected. Optimized code for single-byte character set conversion.
- Implemented `ResultSet.getCharacterStream()`.
- Added `LOCAL TEMPORARY` to table types in `DatabaseMetaData.getTableTypes()`.
- Massive code clean-up to follow Java coding conventions (the time had come).

C.5.4.18. Changes in MySQL Connector/J 3.0.0-dev (31 July 2002)

- **!!! LICENSE CHANGE !!!** The driver is now GPL. If you need non-GPL licenses, please contact me [<mark@mysql.com>](mailto:mark@mysql.com).
- JDBC-3.0 functionality including `Statement/PreparedStatement.getGeneratedKeys()` and `ResultSet.getURL()`.
- Performance enhancements: Driver is now 50–100% faster in most situations, and creates fewer temporary objects.
- Repackaging: New driver name is `com.mysql.jdbc.Driver`, old name still works, though (the driver is now provided by MySQL-AB).
- Better checking for closed connections in `Statement` and `PreparedStatement`.
- Support for streaming (row-by-row) result sets (see [README](#)) Thanks to Doron.
- Support for large packets (new addition to MySQL-4.0 protocol), see [README](#) for more information.
- JDBC Compliance: Passes all tests besides stored procedure tests.
- Fix and sort primary key names in `DBMetaData` (SF bugs 582086 and 582086).
- Float types now reported as `java.sql.Types.FLOAT` (SF bug 579573).
- `ResultSet.getTimestamp()` now works for `DATE` types (SF bug 559134).
- `ResultSet.getDate/Time/Timestamp` now recognizes all forms of invalid values that have been set to all zeros by MySQL (SF bug 586058).
- Testsuite now uses Junit (which you can get from <http://www.junit.org>).
- The driver now only works with JDK-1.2 or newer.
- Added multi-host failover support (see [README](#)).
- General source-code cleanup.
- Overall speed improvements via controlling transient object creation in `MysqlLIO` class when reading packets.
- Performance improvements in string handling and field metadata creation (lazily instantiated) contributed by Alex Twisleton-Wykeham-Fiennes.

C.5.5. Changes in MySQL Connector/J 2.0.x

C.5.5.1. Changes in MySQL Connector/J 2.0.14 (16 May 2002)

- More code cleanup.
- `PreparedStatement` now releases resources on `.close()`. (SF bug 553268)

- Quoted identifiers not used if server version does not support them. Also, if server started with `--ansi` or `--sql-mode=ANSI_QUOTES`, `'` will be used as an identifier quote character, otherwise ``` will be used.
- `ResultSet.getDouble()` now uses code built into JDK to be more precise (but slower).
- `LogicalHandle.isClosed()` calls through to physical connection.
- Added SQL profiling (to `STDERR`). Set `profileSql=true` in your JDBC URL. See [README](#) for more information.
- Fixed typo for `relaxAutoCommit` parameter.

C.5.5.2. Changes in MySQL Connector/J 2.0.13 (24 April 2002)

- More code cleanup.
- Fixed unicode chars being read incorrectly. (SF bug 541088)
- Faster blob escaping for `PreparedStatement`.
- Added `set/getPortNumber()` to `DataSource(s)`. (SF bug 548167)
- Added `setURL()` to `MySQLXADataSource`. (SF bug 546019)
- `PreparedStatement.toString()` fixed. (SF bug 534026)
- `ResultSetMetaData.getColumnClassName()` now implemented.
- Rudimentary version of `Statement.getGeneratedKeys()` from JDBC-3.0 now implemented (you need to be using JDK-1.4 for this to work, I believe).
- `DBMetaData.getIndexInfo()` - bad PAGES fixed. (SF BUG 542201)

C.5.5.3. Changes in MySQL Connector/J 2.0.12 (07 April 2002)

- General code cleanup.
- Added `getIdleFor()` method to `Connection` and `MySQLLogicalHandle`.
- Relaxed synchronization in all classes, should fix 520615 and 520393.
- Added `getTable/ColumnPrivileges()` to DBMD (fixes 484502).
- Added new types to `getTypeInfo()`, fixed existing types thanks to Al Davis and Kid Kalanon.
- Added support for `BIT` types (51870) to `PreparedStatement`.
- Fixed `getRow()` bug (527165) in `ResultSet`.
- Fixes for `ResultSet` updatability in `PreparedStatement`.
- Fixed time zone off-by-1-hour bug in `PreparedStatement` (538286, 528785).
- `ResultSet`: Fixed updatability (values being set to `null` if not updated).
- `DataSources` - fixed `setUrl` bug (511614, 525565), wrong datasource class name (532816, 528767).
- Added identifier quoting to all `DatabaseMetaData` methods that need them (should fix 518108).
- Added support for `YEAR` type (533556).
- `ResultSet.insertRow()` should now detect `auto_increment` fields in most cases and use that value in the new row. This detection will not work in multi-valued keys, however, due to the fact that the MySQL protocol does not return this information.

- `ResultSet.refreshRow()` implemented.
- Fixed `testsuite.Traversal afterLast()` bug, thanks to Igor Lastric.

C.5.5.4. Changes in MySQL Connector/J 2.0.11 (27 January 2002)

- Fixed missing `DELETE_RULE` value in `DBMD.getImported/ExportedKeys()` and `getCrossReference()`.
- Full synchronization of `Statement.java`.
- More changes to fix `Unexpected end of input stream` errors when reading `BLOB` values. This should be the last fix.

C.5.5.5. Changes in MySQL Connector/J 2.0.10 (24 January 2002)

- Fixed spurious `Unexpected end of input stream` errors in `MysqlIO` (bug 507456).
- Fixed null-pointer-exceptions when using `MysqlConnectionPoolDataSource` with Websphere 4 (bug 505839).

C.5.5.6. Changes in MySQL Connector/J 2.0.9 (13 January 2002)

- `Ant` build was corrupting included `jar` files, fixed (bug 487669).
- Fixed extra memory allocation in `MysqlIO.readPacket()` (bug 488663).
- Implementation of `DatabaseMetaData.getExported/ImportedKeys()` and `getCrossReference()`.
- Full synchronization on methods modifying instance and class-shared references, driver should be entirely thread-safe now (please let me know if you have problems).
- `DataSource` implementations moved to `org.gjt.mm.mysql.jdbc2.optional` package, and (initial) implementations of `PooledConnectionDataSource` and `XADataSource` are in place (thanks to Todd Wolff for the implementation and testing of `PooledConnectionDataSource` with IBM WebSphere 4).
- Added detection of network connection being closed when reading packets (thanks to Todd Lizambri).
- Fixed quoting error with escape processor (bug 486265).
- Report batch update support through `DatabaseMetaData` (bug 495101).
- Fixed off-by-one-hour error in `PreparedStatement.setTimestamp()` (bug 491577).
- Removed concatenation support from driver (the `||` operator), as older versions of VisualAge seem to be the only thing that use it, and it conflicts with the logical `||` operator. You will need to start `mysqld` with the `--ansi` flag to use the `||` operator as concatenation (bug 491680).
- Fixed casting bug in `PreparedStatement` (bug 488663).

C.5.5.7. Changes in MySQL Connector/J 2.0.8 (25 November 2001)

- Batch updates now supported (thanks to some inspiration from Daniel Rall).
- `XADataSource/ConnectionPoolDataSource` code (experimental)
- `PreparedStatement.setAnyNumericType()` now handles positive exponents correctly (adds `+` so MySQL can understand it).
- `DatabaseMetaData.getPrimaryKeys()` and `getBestRowIdentifier()` are now more robust in identifying primary keys (matches regardless of case or abbreviation/full spelling of `Primary Key` in `Key_type` column).

C.5.5.8. Changes in MySQL Connector/J 2.0.7 (24 October 2001)

- `PreparedStatement.setCharacterStream()` now implemented
- Fixed dangling socket problem when in high availability (`autoReconnect=true`) mode, and finalizer for `Connection` will close any dangling sockets on GC.
- Fixed `ResultSetMetaData.getPrecision()` returning one less than actual on newer versions of MySQL.
- `ResultSet.getBlob()` now returns `null` if column value was `null`.
- Character sets read from database if `useUnicode=true` and `characterEncoding` is not set. (thanks to Dmitry Vereshchagin)
- Initial transaction isolation level read from database (if available). (thanks to Dmitry Vereshchagin)
- Fixed `DatabaseMetaData.supportsTransactions()`, and `supportsTransactionIsolationLevel()` and `getTypeInfo()` `SQL_DATETIME_SUB` and `SQL_DATA_TYPE` fields not being readable.
- Fixed `PreparedStatement` generating SQL that would end up with syntax errors for some queries.
- Fixed `ResultSet.isAfterLast()` always returning `false`.
- Fixed time zone issue in `PreparedStatement.setTimestamp()`. (thanks to Erik Olofsson)
- Capitalize type names when `capitalizeTypeNames=true` is passed in URL or properties (for WebObjects. (thanks to Anjo Krank)
- Updatable result sets now correctly handle `NULL` values in fields.
- `PreparedStatement.setDouble()` now uses full-precision doubles (reverting a fix made earlier to truncate them).
- `PreparedStatement.setBoolean()` will use 1/0 for values if your MySQL version is 3.21.23 or higher.

C.5.5.9. Changes in MySQL Connector/J 2.0.6 (16 June 2001)

- Fixed `PreparedStatement` parameter checking.
- Fixed case-sensitive column names in `ResultSet.java`.

C.5.5.10. Changes in MySQL Connector/J 2.0.5 (13 June 2001)

- Fixed `ResultSet.getBlob()` `ArrayIndex` out-of-bounds.
- Fixed `ResultSetMetaData.getColumnTypeName` for `TEXT/BLOB`.
- Fixed `ArrayIndexOutOfBoundsException` when sending large `BLOB` queries. (Max size packet was not being set)
- Added `ISOLATION` level support to `Connection.setIsolationLevel()`
- Fixed NPE on `PreparedStatement.executeUpdate()` when all columns have not been set.
- Fixed data parsing of `TIMESTAMP` values with 2-digit years.
- Added `Byte` to `PreparedStatement.setObject()`.
- `ResultSet.getBoolean()` now recognizes `-1` as `true`.
- `ResultSet` has `+/-Inf/inf` support.
- `ResultSet.insertRow()` works now, even if not all columns are set (they will be set to `NULL`).

- `DataBaseMetaData.getCrossReference()` no longer `ArrayIndex00B`.
- `getObject()` on `ResultSet` correctly does `TINYINT->Byte` and `SMALLINT->Short`.

C.5.5.11. Changes in MySQL Connector/J 2.0.3 (03 December 2000)

- Implemented `getBigDecimal()` without scale component for JDBC2.
- Fixed composite key problem with updatable result sets.
- Added detection of `-/+INF` for doubles.
- Faster ASCII string operations.
- Fixed incorrect detection of `MAX_ALLOWED_PACKET`, so sending large blobs should work now.
- Fixed off-by-one error in `java.sql.Blob` implementation code.
- Added `ultraDevHack` URL parameter, set to `true` to allow (broken) Macromedia UltraDev to use the driver.

C.5.5.12. Changes in MySQL Connector/J 2.0.1 (06 April 2000)

- Fixed `RSMD.isWritable()` returning wrong value. Thanks to Moritz Maass.
- Cleaned up exception handling when driver connects.
- Columns that are of type `TEXT` now return as `Strings` when you use `getObject()`.
- `DatabaseMetaData.getPrimaryKeys()` now works correctly with respect to `key_seq`. Thanks to Brian Slesinsky.
- No escape processing is done on `PreparedStatement` anymore per JDBC spec.
- Fixed many JDBC-2.0 traversal, positioning bugs, especially with respect to empty result sets. Thanks to Ron Smits, Nick Brook, Cesar Garcia and Carlos Martinez.
- Fixed some issues with updatability support in `ResultSet` when using multiple primary keys.

C.5.5.13. Changes in MySQL Connector/J 2.0.0pre5 (21 February 2000)

- Fixed Bad Handshake problem.

C.5.5.14. Changes in MySQL Connector/J 2.0.0pre4 (10 January 2000)

- Fixes to `ResultSet` for `insertRow()` - Thanks to Cesar Garcia
- Fix to Driver to recognize JDBC-2.0 by loading a JDBC-2.0 class, instead of relying on JDK version numbers. Thanks to John Baker.
- Fixed `ResultSet` to return correct row numbers
- `Statement.getUpdateCount()` now returns rows matched, instead of rows actually updated, which is more SQL-92 like.

10-29-99

- `Statement/PreparedStatement.getMoreResults()` bug fixed. Thanks to Noel J. Bergman.

- Added Short as a type to PreparedStatement.setObject(). Thanks to Jeff Crowder
- Driver now automagically configures maximum/preferred packet sizes by querying server.
- Autoreconnect code uses fast ping command if server supports it.
- Fixed various bugs with respect to packet sizing when reading from the server and when alloc'ing to write to the server.

C.5.5.15. Changes in MySQL Connector/J 2.0.0pre (17 August 1999)

- Now compiles under JDK-1.2. The driver supports both JDK-1.1 and JDK-1.2 at the same time through a core set of classes. The driver will load the appropriate interface classes at runtime by figuring out which JVM version you are using.
- Fixes for result sets with all nulls in the first row. (Pointed out by Tim Endres)
- Fixes to column numbers in SQLExceptions in ResultSet (Thanks to Blas Rodriguez Somoza)
- The database no longer needs to be specified to connect. (Thanks to Christian Motschke)

C.5.6. Changes in MySQL Connector/J 1.2b (04 July 1999)

- Better Documentation (in progress), in doc/mm.doc/book1.html
- DBMD now allows null for a column name pattern (not in spec), which it changes to '%'
- DBMD now has correct types/lengths for getXXX().
- ResultSet.getDate(), getTime(), and getTimestamp() fixes. (contributed by Alan Wilken)
- EscapeProcessor now handles \{ \} and { or } inside quotes correctly. (thanks to Alik for some ideas on how to fix it)
- Fixes to properties handling in Connection. (contributed by Juho Tikkala)
- ResultSet.getObject() now returns null for NULL columns in the table, rather than bombing out. (thanks to Ben Grosman)
- ResultSet.getObject() now returns Strings for types from MySQL that it doesn't know about. (Suggested by Chris Perdue)
- Removed DataInput/Output streams, not needed, 1/2 number of method calls per IO operation.
- Use default character encoding if one is not specified. This is a work-around for broken JVMs, because according to spec, EVERY JVM must support "ISO8859_1", but they don't.
- Fixed Connection to use the platform character encoding instead of "ISO8859_1" if one isn't explicitly set. This fixes problems people were having loading the character- converter classes that didn't always exist (JVM bug). (thanks to Fritz Elfert for pointing out this problem)
- Changed MySQLIO to re-use packets where possible to reduce memory usage.
- Fixed escape-processor bugs pertaining to { } inside quotes.

C.5.7. Changes in MySQL Connector/J 1.2.x and lower

C.5.7.1. Changes in MySQL Connector/J 1.2a (14 April 1999)

- Fixed character-set support for non-Javasoft JVMs (thanks to many people for pointing it out)
- Fixed ResultSet.getBoolean() to recognize 'y' & 'n' as well as '1' & '0' as boolean flags. (thanks to Tim Pizey)
- Fixed ResultSet.getTimestamp() to give better performance. (thanks to Richard Swift)

- Fixed `getBytes()` for numeric types. (thanks to Ray Bellis)
- Fixed `DatabaseMetaData.getTypeInfo()` for DATE type. (thanks to Paul Johnston)
- Fixed `EscapeProcessor` for "fn" calls. (thanks to Piyush Shah at locomotive.org)
- Fixed `EscapeProcessor` to not do extraneous work if there are no escape codes. (thanks to Ryan Gustafson)
- Fixed `Driver` to parse URLs of the form "jdbc:mysql://host:port" (thanks to Richard Lobb)

C.5.7.2. Changes in MySQL Connector/J 1.1i (24 March 1999)

- Fixed Timestamps for `PreparedStatement`s
- Fixed null pointer exceptions in `RSMD` and `RS`
- Re-compiled with `jikes` for valid class files (thanks ms!)

C.5.7.3. Changes in MySQL Connector/J 1.1h (08 March 1999)

- Fixed escape processor to deal with unmatched { and } (thanks to Craig Coles)
- Fixed escape processor to create more portable (between DATETIME and TIMESTAMP types) representations so that it will work with BETWEEN clauses. (thanks to Craig Longman)
- `MysqlIO.quit()` now closes the socket connection. Before, after many failed connections some OS's would run out of file descriptors. (thanks to Michael Brinkman)
- Fixed `NullPointerException` in `Driver.getPropertyInfo`. (thanks to Dave Potts)
- Fixes to `MysqlDefs` to allow all *text fields to be retrieved as Strings. (thanks to Chris at Leverage)
- Fixed `setDouble` in `PreparedStatement` for large numbers to avoid sending scientific notation to the database. (thanks to J.S. Ferguson)
- Fixed `getScale()` and `getPrecision()` in `RSMD`. (contrib'd by James Klicman)
- Fixed `getObject()` when field was DECIMAL or NUMERIC (thanks to Bert Hobbs)
- `DBMD.getTables()` bombed when passed a null table-name pattern. Fixed. (thanks to Richard Lobb)
- Added check for "client not authorized" errors during connect. (thanks to Hannes Wallnoefer)

C.5.7.4. Changes in MySQL Connector/J 1.1g (19 February 1999)

- Result set rows are now byte arrays. Blobs and Unicode work bidirectionally now. The `useUnicode` and `encoding` options are implemented now.
- Fixes to `PreparedStatement` to send binary set by `setXXXStream` to be sent untouched to the MySQL server.
- Fixes to `getDriverPropertyInfo()`.

C.5.7.5. Changes in MySQL Connector/J 1.1f (31 December 1998)

- Changed all `ResultSet` fields to Strings, this should allow Unicode to work, but your JVM must be able to convert between the character sets. This should also make reading data from the server be a bit quicker, because there is now no conversion from `StringBuffer` to String.

- Changed PreparedStatement.streamToString() to be more efficient (code from Uwe Schaefer).
- URL parsing is more robust (throws SQL exceptions on errors rather than NullPointerExceptions)
- PreparedStatement now can convert Strings to Time/Date values via setObject() (code from Robert Currey).
- IO no longer hangs in Buffer.readInt(), that bug was introduced in 1.1d when changing to all byte-arrays for result sets. (Pointed out by Samo Login)

C.5.7.6. Changes in MySQL Connector/J 1.1b (03 November 1998)

- Fixes to DatabaseMetaData to allow both IBM VA and J-Builder to work. Let me know how it goes. (thanks to Jac Kersing)
- Fix to ResultSet.getBoolean() for NULL strings (thanks to Barry Lagerweij)
- Beginning of code cleanup, and formatting. Getting ready to branch this off to a parallel JDBC-2.0 source tree.
- Added "final" modifier to critical sections in MySQLIO and Buffer to allow compiler to inline methods for speed.

9-29-98

- If object references passed to setXXX() in PreparedStatement are null, setNull() is automatically called for you. (Thanks for the suggestion goes to Erik Ostrom)
- setObject() in PreparedStatement will now attempt to write a serialized representation of the object to the database for objects of Types.OTHER and objects of unknown type.
- Util now has a static method readObject() which given a ResultSet and a column index will re-instantiate an object serialized in the above manner.

C.5.7.7. Changes in MySQL Connector/J 1.1 (02 September 1998)

- Got rid of "ugly hack" in MySQLIO.nextRow(). Rather than catch an exception, Buffer.isLastDataPacket() was fixed.
- Connection.getCatalog() and Connection.setCatalog() should work now.
- Statement.setMaxRows() works, as well as setting by property maxRows. Statement.setMaxRows() overrides maxRows set via properties or url parameters.
- Automatic re-connection is available. Because it has to "ping" the database before each query, it is turned off by default. To use it, pass in "autoReconnect=true" in the connection URL. You may also change the number of reconnect tries, and the initial timeout value via "maxReconnects=n" (default 3) and "initialTimeout=n" (seconds, default 2) parameters. The timeout is an exponential backoff type of timeout; for example, if you have initial timeout of 2 seconds, and maxReconnects of 3, then the driver will timeout 2 seconds, 4 seconds, then 16 seconds between each re-connection attempt.

C.5.7.8. Changes in MySQL Connector/J 1.0 (24 August 1998)

- Fixed handling of blob data in Buffer.java
- Fixed bug with authentication packet being sized too small.
- The JDBC Driver is now under the LPGL

8-14-98

- Fixed Buffer.readLenString() to correctly read data for BLOBS.

- Fixed `PreparedStatement.toString()` to correctly read data for BLOBS.
- Fixed `PreparedStatement.setDate()` to not add a day. (above fixes thanks to Vincent Partington)
- Added URL parameter parsing (`?user=...` and so forth).

C.5.7.9. Changes in MySQL Connector/J 0.9d (04 August 1998)

- Big news! New package name. Tim Endres from ICE Engineering is starting a new source tree for GNU GPL'd Java software. He's graciously given me the `org.gjt.mm` package directory to use, so now the driver is in the `org.gjt.mm.mysql` package scheme. I'm "legal" now. Look for more information on Tim's project soon.
- Now using dynamically sized packets to reduce memory usage when sending commands to the DB.
- Small fixes to `getTypeInfo()` for parameters, and so forth.
- `DatabaseMetaData` is now fully implemented. Let me know if these drivers work with the various IDEs out there. I've heard that they're working with JBuilder right now.
- Added JavaDoc documentation to the package.
- Package now available in `.zip` or `.tar.gz`.

C.5.7.10. Changes in MySQL Connector/J 0.9 (28 July 1998)

- Implemented `getTypeInfo()`. `Connection.rollback()` now throws an `SQLException` per the JDBC spec.
- Added `PreparedStatement` that supports all JDBC API methods for `PreparedStatement` including `InputStreams`. Please check this out and let me know if anything is broken.
- Fixed a bug in `ResultSet` that would break some queries that only returned 1 row.
- Fixed bugs in `DatabaseMetaData.getTables()`, `DatabaseMetaData.getColumns()` and `DatabaseMetaData.getCatalogs()`.
- Added functionality to `Statement` that allows `executeUpdate()` to store values for IDs that are automatically generated for `AUTO_INCREMENT` fields. Basically, after an `executeUpdate()`, look at the `SQLWarnings` for warnings like `"LAST_INSERTED_ID = 'some number', COMMAND = 'your SQL query'"`. If you are using `AUTO_INCREMENT` fields in your tables and are executing a lot of `executeUpdate()`s on one `Statement`, be sure to `clearWarnings()` every so often to save memory.

C.5.7.11. Changes in MySQL Connector/J 0.8 (06 July 1998)

- Split `MysqlIO` and `Buffer` to separate classes. Some `ClassLoaders` gave an `IllegalAccessException` error for some fields in those two classes. Now `mm.mysql` works in applets and all classloaders. Thanks to Joe Ennis <jce@mail.boone.com> for pointing out the problem and working on a fix with me.

C.5.7.12. Changes in MySQL Connector/J 0.7 (01 July 1998)

- Fixed `DatabaseMetadata` problems in `getColumns()` and bug in switch statement in the `Field` constructor. Thanks to Costin Manolache <costin@tdiinc.com> for pointing these out.

C.5.7.13. Changes in MySQL Connector/J 0.6 (21 May 1998)

- Incorporated efficiency changes from Richard Swift <Richard.Swift@kanatek.ca> in `MysqlIO.java` and `ResultSet.java`:
- We're now 15% faster than gwe's driver.

- Started working on `DatabaseMetaData`.
- The following methods are implemented:
 - `getTables()`
 - `getTableTypes()`
 - `getColumns`
 - `getCatalogs()`

C.6. MySQL Connector/MXJ Change History

C.6.1. Changes in MySQL Connector/MXJ 5.0.6 (04 May 2007)

- Removed `use-default-architecture` property replaced.
- Moved `platform-map.properties` into `db-files.jar`.
- Updated `build.xml` in preparation for next beta build.
- Clarified the startup max wait numbers.
- Changed tests to shutdown mysqld prior to deleting files.
- Delete `portFile` on shutdown.
- Swapped out commercial binaries for v5.0.40.
- Fixed port file to always be written to datadir.
- Added `os.name-os.arch` to resource directory mapping properties file.

C.6.2. Changes in MySQL Connector/MXJ 5.0.5 (14 March 2007)

Changes in this release:

- Updated `build.xml` in prep for next release.
- Swapped out gpl binaries for v5.0.37.
- Swapped out commercial binaries for v5.0.36.
- Found and removed dynamic linking in `mysql_kill`; updated solution.
- Changed `connector-mxj.properties` default mysql version to 5.0.37.
- Replaced `Boolean.parseBoolean` with JDK 1.4 compliant `valueOf`.
- Added Patched `StandardSocketFactory` from Connector/J 5-0 HEAD.
- `build.xml: usage` now slightly more verbose; some reformatting.
- Replaced windows `kill.exe` resource with re-written version specific to mysqld.
- `SIGHUP` is replaced with `MySQLShutdown<PID>` event.
- Now incorporates Reggie Burnett's `SafeTerminateProcess` and only calls the unsafe `TerminateProcess` as a final last resort.
- New windows `kill.exe` fixes bug where mysqld was being force terminated. Issue reported by bruno haleblan and others, see:

[MySQL Forums](#).

- In testing so far `mysqld` reliably shuts down cleanly much faster.
- Changed protected constructor of `SimpleMysqldDynamicMBean` from taking a `MysqldResource` to taking a `MysqldFactory`, in order to lay groundwork for addressing BUG discovered by Andrew Rubinger. See: [MySQL Forums](#) (Actual testing with JBoss, and filing a bug, is still required.)
- Changed `SimpleMysqldDynamicMBean` to create `MysqldResource` on demand in order to allow setting of `datadir`. (Rubinger bug groundwork).
- Added `getDataDir()` to interface `MysqldResourceI`.
- Added testcase to `com.mysql.management.jmx.AcceptanceTest` which demonstrates that `dataDir` is a mutable MBean property.
- Moved `MysqldFactory` to main package.
- Reformatting: Added newlines some files which did not end in them.
- Added 5.1.15 binaries to the repository.
- Removed 5.1.14 binaries from the repository.
- Clarified the synchronization of `MysqldResource` methods.
- Clarified the immutability of `baseDir`, `dataDir`, `pidFile`, `portFile`.
- Removed 5.0.22 binaries from the repository.
- Added 5.1.14 binaries to repository.
- Ensured 5.1.14 compatibility.

C.6.3. Changes in MySQL Connector/MXJ 5.0.4 (28 January 2007)

- Swapped out gpl binaries for v5.0.27.
- Swapped out commercial binaries for v5.0.32.
- Updated `build.xml` to build to handle with different gpl and commercial `mysld` version numbers.
- Moved `mysqld` binary resourced into separate jar file NOTICE: `CLASSPATH` will now need to `connector-mxj-db-files.jar`.
- Minor test robustness improvements.
- Moved default version string out of java class into a text editable properties file (`connector-mxj.properties`) in the resources directory.
- Introduced property for Linux & WinXX to default to 32bit versions.
- Allow multiple calls to start server from URL connection on non-3306 port. ([Bug#24004](#))
- Only populate the options map from the help text if specifically requested or in the MBean case.
- Fixed test to be tollerant of `/tmp` being a symlink to `/foo/tmp`.

C.6.4. Changes in MySQL Connector/MXJ 5.0.3 (24 June 2006)

- Swapped out the `mysqld` binaries for MySQL v5.0.22.

- Removed "TeeOutputStream" - no longer needed.
- Removed unused imports, formatted code, made minor edits to tests.

C.6.5. Changes in MySQL Connector/MXJ 5.0.2 (15 June 2006)

Changes in this release:

- Replaced string parsing with JDBC connection attempt for determining if a mysqld is "ready for connections" `CLASSPATH` will now need to include Connector/J jar.
- Extended timeout for help string parsing, to avoid cases where the help text was getting prematurely flushed, and thus truncated.

Changes made on 11 May 2006:

- Added trace level logging with Aspect/J. `CLASSPATH` will now need to include `lib/aspectjrt.jar`

Changes made on 11 May 2006:

- Swapped out the mysqld binaries for MySQL v5.0.21

Changes made on 04 May 2006:

- `ServerLauncherSocketFactory` now treats URL parameters in the form of `&server.foo=null` as `serverOption-Map.put("foo", null)`

Changes made on 26 April 2006:

- `MysqldResource` now tied to `dataDir` as well as `basedir` (API CHANGE)
- moved PID file into `datadir`
- `ServerLauncherSocketFactory.shutdown` API change: now takes 2 File parameters (`basedir`, `datadir`)
- insulated users from problems with "." in `basedir`

Changes made on 25 April 2006:

- Made tests more robust by deleting the `/tmp/test-c.mxj` directory before running tests.
- socket is now "mysql.sock" in `datadir`
- `ServerLauncherSocketFactory.shutdown` now works across JVMs.
- `ServerLauncherSocketFactory.shutdown` API change: now takes File parameter (`basedir`) instead of port.
- altered to be "basedir" rather than "port" oriented.

Changes made on 18 April 2006:

- `ServerLauncherSocketFactory.shutdown(port)` no longer throws, only reports to `System.err`

Changes made on 16 March 2006:

- swapped out the mysqld binaries for MySQL v5.0.19
- added ability to specify "mysql-version" as an url parameter
- extracted splitLines(String) to Str utility class
- extracted array and list printing to ListToString utility class
- reformatted code

Changes made on 16 January 2006:

- swapped out the mysqld binaries for MySQL v5.0.18
- "platform" directories replace spaces with underscores
- help parsing test reflects current help options

C.6.6. Changes in MySQL Connector/MXJ 5.0.1 (Never released)

This was an internal only release.

C.6.7. Changes in MySQL Connector/MXJ 5.0.0 (09 December 2005)

Changes made on 16 January 2006:

- Swapped out the mysqld binaries for MySQL v5.0.16.

Changes made on 25 October 2005:

- Altered examples and tests to use new Connector/J 5.0 URL syntax for for launching Connector/MXJ ("jdbc:mysql:mxj://")
- Minor refactorings for type casting and exception handling.

Changes made on 30 August 2005:

- Reorganized utils into a single "Utils" collaborator.
- Ditched "ClassUtil" (merged with Str).
- Removed HelpOptionsParser's need to reference a MysqldResource.

Changes made on 29 August 2005:

- Minor test tweaks

C.7. MySQL Proxy Change History

C.7.1. Changes in MySQL MySQL Proxy 0.6.0 (Not yet released)

Functionality added or changed:

- Added `--no-daemon` and `--pid-file`.
 - Added `--no-proxy` to disable the proxy.
 - Added testcases.
 - Added support for `proxy.response.packets`.
 - Added hooks for `read_auth()`, `read_handshake()` and `read_auth_result()`.
 - Added support in `read_query_result()` to overwrite the result-set.
 - Added connection pooling.
 - Added a global lua-scope `proxy.global.*`.
 - Added support for listening UNIX sockets.
 - Added handling of `proxy.connection.backend_ndx` in `connect_server()` and `read_query()` to support read/write splitting.
-
- Fixed decoding of len-encoded ints for 3-byte notation.
 - Fixed check for `glib2` to require at least 2.6.0.
 - Fixed mem-leak with `proxy.response.*` is used.
 - Fixed handling of (SQL) `NULL` in result-sets.
 - Fixed `inj.resultset.affected_rows` on `SELECT` queries.
 - Fixed len-encoding on `proxy.resultsets`.
 - Fixed assertion when all backends are down and we try to connect.
 - Fixed assertion when connecting to the MySQL 6.0.1.
 - Fixed crash if `proxy.connection` is used in `connect_server()`.
 - Fixed connection-stalling if `read_query_result()` throws an `assert()`ion.
 - Fixed assertion at `COM_SHUTDOWN`. (Bug#29719)
 - Fixed assertion at login with empty password + empty default db. (Bug#29719)
 - Fixed assertion on result-packets like `[ field-len | fields | EOF | ERR ]`. (Bug#29732)
 - Fixed compilation on win32.
 - Fixed assertion on `COM_BINLOG_DUMP`. (Bug#29764)

C.7.2. Changes in MySQL MySQL Proxy 0.5.1 (30 June 2007)

Functionality added or changed:

- Added script examples for rewriting and injection.
- Added support for UNIX sockets.
- Added protection against duplicate resultsets from a script.
- Added missing dependency to `libmysqlclient-dev` to the `INSTALL` file.

- Added support for pre-4.1 passwords in a 4.1 connection.
- Added `inj.query_time` and `inj.response_time` into the lua scripts.
- Added `resultset.affected_rows` and `resultset.insert_id`.
- Added `proxy.VERSION`.
- Changed `--proxy.profiling` to `--proxy-skip-profiling`.

Bugs fixed:

- Fixed assertion when `read_query_result()` is not provided when `PROXY_SEND_QUERY` is used.
- Fixed warning if `connect_server()` is not provided.
- Fixed handling of duplicate ERR on `COM_CHANGE_USER` in MySQL 5.1.18+.
- Fixed compile error with MySQL 4.1.x on missing `COM_STMT_*`.
- Fixed mysql check in configure to die when `mysql.h` isn't detected.
- Fixed crash on fields > 250 bytes when the resultset is inspected.
- Fixed assertion when a error occurs at initial script exec time.

C.7.3. Changes in MySQL MySQL Proxy 0.5.0 (19 June 2007)

This is the first beta release.

Features in this release:

- Added `automake/autoconf` support.
- Added `cmake` support.

Appendix D. Limits and Restrictions

The discussion here describes restrictions that apply to the use of MySQL features such as subqueries or views.

D.1. Restrictions on Stored Routines and Triggers

Some of the restrictions noted here apply to all stored routines; that is, both to stored procedures and stored functions. Some of restrictions apply only to stored functions, and not to stored procedures.

All of the restrictions for stored functions also apply to triggers.

Stored routines cannot contain arbitrary SQL statements. The following statements are disallowed:

- The locking statements `LOCK TABLES`, `UNLOCK TABLES`.
- `LOAD DATA` and `LOAD TABLE`.
- SQL prepared statements (`PREPARE`, `EXECUTE`, `DEALLOCATE PREPARE`). Implication: You cannot use dynamic SQL within stored functions or triggers (where you construct dynamically statements as strings and then execute them).

In addition, SQL statements that are not permitted within prepared statements are also not permitted in stored routines. See [Section 12.7, “SQL Syntax for Prepared Statements”](#), for a list of statements supported in prepared statements. Statements not listed there are not supported for SQL prepared statements and thus are also not supported for stored routines unless noted otherwise in [Chapter 19, *Stored Procedures and Functions*](#).

For stored functions (but not stored procedures), the following additional statements or operations are disallowed:

- Statements that do explicit or implicit commit or rollback.
- Statements that return a result set. This includes `SELECT` statements that do not have an `INTO var_list` clause and `SHOW` statements. A function can process a result set either with `SELECT ... INTO var_list` or by using a cursor and `FETCH` statements. See [Section 19.2.7.3, “SELECT ... INTO Statement”](#).
- `FLUSH` statements.
- Recursive statements. That is, stored functions cannot be used recursively.
- Within a stored function or trigger, it is not permitted to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.
- `ALTER VIEW`.

Note that although some restrictions normally apply to stored functions and triggers but not to stored procedures, those restrictions do apply to stored procedures if they are invoked from within a stored function or trigger. For example, although you can use `FLUSH` in a stored procedure, such a stored procedure cannot be called from a stored function or trigger.

It is possible for the same identifier to be used for a routine parameter, a local variable, and a table column. Also, the same local variable name can be used in nested blocks. For example:

```
CREATE PROCEDURE p (i INT)
BEGIN
  DECLARE i INT DEFAULT 0;
  SELECT i FROM t;
  BEGIN
    DECLARE i INT DEFAULT 1;
    SELECT i FROM t;
  END;
END;
```

In such cases the identifier is ambiguous and the following precedence rules apply:

- A local variable takes precedence over a routine parameter or table column
- A routine parameter takes precedence over a table column
- A local variable in an inner block takes precedence over a local variable in an outer block

The behavior that table columns do not take precedence over variables is non-standard.

Use of stored routines can cause replication problems. This issue is discussed further in [Section 19.4, “Binary Logging of Stored Routines and Triggers”](#).

`INFORMATION_SCHEMA` does not yet have a `PARAMETERS` table, so applications that need to acquire routine parameter information at runtime must use workarounds such as parsing the output of `SHOW CREATE` statements.

There are no stored routine debugging facilities.

Before MySQL 5.1.4, `CALL` statements cannot be prepared. This true both for server-side prepared statements and for SQL prepared statements.

`UNDO` handlers are not supported.

`FOR` loops are not supported.

To prevent problems of interaction between server threads, when a client issues a statement, the server uses a snapshot of routines and triggers available for execution of the statement. That is, the server calculates a list of procedures, functions, and triggers that may be used during execution of the statement, loads them, and then proceeds to execute the statement. This means that while the statement executes, it will not see changes to routines performed by other threads.

The `RENAME DATABASE` statement does not migrate stored routines to the new schema name. See [Section 12.1.18, “RENAME DATABASE Syntax”](#).

For triggers, the following additional statements or operations are disallowed:

- Triggers currently are not activated by foreign key actions.
- The `RETURN` statement is disallowed in triggers, which cannot return a value. To exit a trigger immediately, use the `LEAVE` statement.
- Triggers are not allowed on tables in the `mysql` database.

Stored routines and triggers in MySQL Cluster. Stored functions, stored procedures, and triggers are all supported by tables using the `NDB` storage engine; however, it is important to keep in mind that they do *not* propagate automatically between MySQL Servers acting as Cluster SQL nodes. This is because of the following:

- Stored routine definitions are kept in tables in the `mysql` system database using the `MyISAM` storage engine, and so do not participate in clustering.
- The `.TRN` and `.TRG` files containing trigger definitions are not read by the `NDB` storage engine, and are not copied between Cluster nodes.

Any stored routine or trigger that interacts with MySQL Cluster tables must be re-created by running the appropriate `CREATE PROCEDURE`, `CREATE FUNCTION`, or `CREATE TRIGGER` statements on each MySQL Server that participates in the cluster where you wish to use the stored routine or trigger. Similarly, any changes to existing stored routines or triggers must be carried out explicitly on all Cluster SQL nodes, using the appropriate `ALTER` or `DROP` statements on each MySQL Server accessing the cluster.

Warning

Do *not* attempt to work around the issue described in the first item mentioned previously by converting any `mysql` database tables to use the `NDB` storage engine. *Altering the system tables in the `mysql` database is very likely to produce undesirable results, and is not supported by MySQL AB.*

D.2. Restrictions on Server-Side Cursors

Server-side cursors are implemented in the C API via the `mysql_stmt_attr_set()` function. The same implementation is used for cursors in stored routines. A server-side cursor allows a result set to be generated on the server side, but not transferred to the client except for those rows that the client requests. For example, if a client executes a query but is only interested in the first row, the remaining rows are not transferred.

In MySQL, a server-side cursor is materialized into a temporary table. Initially, this is a **MEMORY** table, but is converted to a **MyISAM** table if its size reaches the value of the `max_heap_table_size` system variable. One limitation of the implementation is that for a large result set, retrieving its rows through a cursor might be slow.

Cursors are read-only; you cannot use a cursor to update rows.

UPDATE WHERE CURRENT OF and **DELETE WHERE CURRENT OF** are not implemented, because updatable cursors are not supported.

Cursors are non-holdable (not held open after a commit).

Cursors are asensitive.

Cursors are non-scrollable.

Cursors are not named. The statement handler acts as the cursor ID.

You can have open only a single cursor per prepared statement. If you need several cursors, you must prepare several statements.

You cannot use a cursor for a statement that generates a result set if the statement is not supported in prepared mode. This includes statements such as **CHECK TABLES**, **HANDLER READ**, and **SHOW BINLOG EVENTS**.

D.3. Restrictions on Subqueries

- In MySQL 5.1 before 5.1.16, if you compare a **NULL** value to a subquery using **ALL**, **ANY**, or **SOME**, and the subquery returns an empty result, the comparison might evaluate to the non-standard result of **NULL** rather than to **TRUE** or **FALSE**.
- A subquery's outer statement can be any one of: **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **SET**, or **DO**.
- Subquery optimization for **IN** is not as effective as for the **=** operator or for **IN(value_list)** constructs.

A typical case for poor **IN** subquery performance is when the subquery returns a small number of rows but the outer query returns a large number of rows to be compared to the subquery result.

The problem is that, for a statement that uses an **IN** subquery, the optimizer rewrites it as a correlated subquery. Consider the following statement that uses an uncorrelated subquery:

```
SELECT ... FROM t1 WHERE t1.a IN (SELECT b FROM t2);
```

The optimizer rewrites the statement to a correlated subquery:

```
SELECT ... FROM t1 WHERE EXISTS (SELECT 1 FROM t2 WHERE t2.b = t1.a);
```

If the inner and outer queries return *M* and *N* rows, respectively, the execution time becomes on the order of $O(M \times N)$, rather than $O(M+N)$ as it would be for an uncorrelated subquery.

An implication is that an **IN** subquery can be much slower than a query written using an **IN(value_list)** construct that lists the same values that the subquery would return.

- In general, you cannot modify a table and select from the same table in a subquery. For example, this limitation applies to statements of the following forms:

```
DELETE FROM t WHERE ... (SELECT ... FROM t ...);  
UPDATE t ... WHERE col = (SELECT ... FROM t ...);  
{INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
```

Exception: The preceding prohibition does not apply if you are using a subquery for the modified table in the **FROM** clause. Ex-

ample:

```
UPDATE t ... WHERE col = (SELECT (SELECT ... FROM t...) AS _t ...);
```

Here the prohibition does not apply because the result from a subquery in the **FROM** clause is stored as a temporary table, so the relevant rows in **t** have already been selected by the time the update to **t** takes place.

- Row comparison operations are only partially supported:
 - For **expr IN (subquery)**, **expr** can be an *n*-tuple (specified via row constructor syntax) and the subquery can return rows of *n*-tuples.
 - For **expr op {ALL|ANY|SOME} (subquery)**, **expr** must be a scalar value and the subquery must be a column subquery; it cannot return multiple-column rows.

In other words, for a subquery that returns rows of *n*-tuples, this is supported:

```
(val_1, ..., val_n) IN (subquery)
```

But this is not supported:

```
(val_1, ..., val_n) op {ALL|ANY|SOME} (subquery)
```

The reason for supporting row comparisons for **IN** but not for the others is that **IN** is implemented by rewriting it as a sequence of **=** comparisons and **AND** operations. This approach cannot be used for **ALL**, **ANY**, or **SOME**.

- Row constructors are not well optimized. The following two expressions are equivalent, but only the second can be optimized:

```
(col1, col2, ...) = (val1, val2, ...)
col1 = val1 AND col2 = val2 AND ...
```

- Subqueries in the **FROM** clause cannot be correlated subqueries. They are materialized (executed to produce a result set) before evaluating the outer query, so they cannot be evaluated per row of the outer query.
- The optimizer is more mature for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as a join.

An exception occurs for the case where an **IN** subquery can be rewritten as a **SELECT DISTINCT** join. Example:

```
SELECT col FROM t1 WHERE id_col IN (SELECT id_col2 FROM t2 WHERE condition);
```

That statement can be rewritten as follows:

```
SELECT DISTINCT col FROM t1, t2 WHERE t1.id_col = t2.id_col AND condition;
```

But in this case, the join requires an extra **DISTINCT** operation and is not more efficient than the subquery.

- Possible future optimization: MySQL does not rewrite the join order for subquery evaluation. In some cases, a subquery could be executed more efficiently if MySQL rewrote it as a join. This would give the optimizer a chance to choose between more execution plans. For example, it could decide whether to read one table or the other first.

Example:

```
SELECT a FROM outer_table AS ot
WHERE a IN (SELECT a FROM inner_table AS it WHERE ot.b = it.b);
```

For that query, MySQL always scans **outer_table** first and then executes the subquery on **inner_table** for each row. If **outer_table** has a lot of rows and **inner_table** has few rows, the query probably will not be as fast as it could be.

The preceding query could be rewritten like this:

```
SELECT a FROM outer_table AS ot, inner_table AS it
WHERE ot.a = it.a AND ot.b = it.b;
```

In this case, we can scan the small table (`inner_table`) and look up rows in `outer_table`, which will be fast if there is an index on (`ot.a`, `ot.b`).

- Possible future optimization: A correlated subquery is evaluated for each row of the outer query. A better approach is that if the outer row values do not change from the previous row, do not evaluate the subquery again. Instead, use its previous result.
- Possible future optimization: A subquery in the `FROM` clause is evaluated by materializing the result into a temporary table, and this table does not use indexes. This does not allow the use of indexes in comparison with other tables in the query, although that might be useful.
- Possible future optimization: If a subquery in the `FROM` clause resembles a view to which the merge algorithm can be applied, rewrite the query and apply the merge algorithm so that indexes can be used. The following statement contains such a subquery:

```
SELECT * FROM (SELECT * FROM t1 WHERE t1.t1_col)
           AS _t1, t2 WHERE t2.t2_col;
```

The statement can be rewritten as a join like this:

```
SELECT * FROM t1, t2 WHERE t1.t1_col AND t2.t2_col;
```

This type of rewriting would provide two benefits:

- It avoids the use of a temporary table for which no indexes can be used. In the rewritten query, the optimizer can use indexes on `t1`.
- It gives the optimizer more freedom to choose between different execution plans. For example, rewriting the query as a join allows the optimizer to use `t1` or `t2` first.
- Possible future optimization: For `IN`, `= ANY`, `<> ANY`, `= ALL`, and `<> ALL` with uncorrelated subqueries, use an in-memory hash for a result or a temporary table with an index for larger results. Example:

```
SELECT a FROM big_table AS bt
WHERE non_key_field IN (SELECT non_key_field FROM table WHERE condition)
```

In this case, we could create a temporary table:

```
CREATE TABLE t (key (non_key_field))
(SELECT non_key_field FROM table WHERE condition)
```

Then, for each row in `big_table`, do a key lookup in `t` based on `bt.non_key_field`.

D.4. Restrictions on Views

View processing is not optimized:

- It is not possible to create an index on a view.
- Indexes can be used for views processed using the merge algorithm. However, a view that is processed with the temptable algorithm is unable to take advantage of indexes on its underlying tables (although indexes can be used during generation of the temporary tables).

Subqueries cannot be used in the `FROM` clause of a view. This limitation will be lifted in the future.

There is a general principle that you cannot modify a table and select from the same table in a subquery. See [Section D.3, “Restrictions on Subqueries”](#).

The same principle also applies if you select from a view that selects from the table, if the view selects from the table in a subquery and the view is evaluated using the merge algorithm. Example:

```
CREATE VIEW v1 AS
SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);
```

```
UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

If the view is evaluated using a temporary table, you *can* select from the table in the view subquery and still modify that table in the outer query. In this case the view will be stored in a temporary table and thus you are not really selecting from the table in a subquery and modifying it “at the same time.” (This is another reason you might wish to force MySQL to use the temptable algorithm by specifying `ALGORITHM = TEMPTABLE` in the view definition.)

You can use `DROP TABLE` or `ALTER TABLE` to drop or alter a table that is used in a view definition (which invalidates the view) and no warning results from the drop or alter operation. An error occurs later when the view is used.

A view definition is “frozen” by certain statements:

- If a statement prepared by `PREPARE` refers to a view, the view contents seen each time the statement is executed later will be the contents of the view at the time it was prepared. This is true even if the view definition is changed after the statement is prepared and before it is executed. Example:

```
CREATE VIEW v AS SELECT 1;
PREPARE s FROM 'SELECT * FROM v';
ALTER VIEW v AS SELECT 2;
EXECUTE s;
```

The result returned by the `EXECUTE` statement is 1, not 2.

- If a statement in a stored routine refers to a view, the view contents seen by the statement are its contents the first time that statement is executed. For example, this means that if the statement is executed in a loop, further iterations of the statement see the same view contents, even if the view definition is changed later in the loop. Example:

```
CREATE VIEW v AS SELECT 1;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  WHILE i < 5 DO
    SELECT * FROM v;
    SET i = i + 1;
    ALTER VIEW v AS SELECT 2;
  END WHILE;
END;
//
delimiter ;
CALL p();
```

When the procedure `p()` is called, the `SELECT` returns 1 each time through the loop, even though the view definition is changed within the loop.

With regard to view updatability, the overall goal for views is that if any view is theoretically updatable, it should be updatable in practice. This includes views that have `UNION` in their definition. Currently, not all views that are theoretically updatable can be updated. The initial view implementation was deliberately written this way to get usable, updatable views into MySQL as quickly as possible. Many theoretically updatable views can be updated now, but limitations still exist:

- Updatable views with subqueries anywhere other than in the `WHERE` clause. Some views that have subqueries in the `SELECT` list may be updatable.
- You cannot use `UPDATE` to update more than one underlying table of a view that is defined as a join.
- You cannot use `DELETE` to update a view that is defined as a join.

There exists a shortcoming with the current implementation of views. If a user is granted the basic privileges necessary to create a view (the `CREATE VIEW` and `SELECT` privileges), that user will be unable to call `SHOW CREATE VIEW` on that object unless the user is also granted the `SHOW VIEW` privilege.

That shortcoming can lead to problems backing up a database with `mysqldump`, which may fail due to insufficient privileges. This problem is described in [Bug#22062](#).

The workaround to the problem is for the administrator to manually grant the `SHOW VIEW` privilege to users who are granted `CREATE VIEW`, since MySQL doesn't grant it implicitly when views are created.

D.5. Restrictions on XA Transactions

XA transaction support is limited to the `InnoDB` storage engine.

The MySQL XA implementation is for “external XA,” where a MySQL server acts as a Resource Manager and client programs act as Transaction Managers. “Internal XA” is not implemented. This would allow individual storage engines within a MySQL server to act as RMs, and the server itself to act as a TM. Internal XA is required for handling XA transactions that involve more than one storage engine. The implementation of internal XA is incomplete because it requires that a storage engine support two-phase commit at the table handler level, and currently this is true only for `InnoDB`.

For `XA START`, the `JOIN` and `RESUME` clauses are not supported.

For `XA END`, the `SUSPEND [FOR MIGRATE]` clause is not supported.

The requirement that the `bqual` part of the `xid` value be different for each XA transaction within a global transaction is a limitation of the current MySQL XA implementation. It is not part of the XA specification.

If an XA transaction has reached the `PREPARED` state and the MySQL server is killed (for example, with `kill -9` on Unix) or shuts down abnormally, the transaction can be continued after the server restarts. However, if the client reconnects and commits the transaction, the transaction will be absent from the binary log even though it has been committed. This means the data and the binary log have gone out of synchrony. An implication is that XA cannot be used safely together with replication.

It is possible that the server will roll back a pending XA transaction, even one that has reached the `PREPARED` state. This happens if a client connection terminates and the server continues to run, or if clients are connected and the server shuts down gracefully. (In the latter case, the server marks each connection to be terminated, and then rolls back the `PREPARED` XA transaction associated with it.) It should be possible to commit or roll back a `PREPARED` XA transaction, but this cannot be done without changes to the binary logging mechanism.

D.6. Limits in MySQL

This section lists current limits in MySQL 5.1.

D.6.1. Limits of Joins

The maximum number of tables that can be referenced in a single join is 61. This also applies to the number of tables that can be referenced in the definition of a view.

Appendix E. Credits

This appendix lists the developers, contributors, and supporters that have helped to make MySQL what it is today.

E.1. Developers at MySQL AB

These are the developers that are or have been employed by MySQL AB to work on the [MySQL](#) database software, roughly in the order they started to work with us. Following each developer is a small list of the tasks that the developer is responsible for, or the accomplishments they have made. All developers are involved in support.

- Michael (Monty) Widenius
 - Lead developer and main author of the MySQL server ([mysqld](#)).
 - New functions for the string library.
 - Most of the [mysys](#) library.
 - The [ISAM](#) and [MyISAM](#) libraries (B-tree index file handlers with index compression and different record formats).
 - The [HEAP](#) library. A memory table system with our superior full dynamic hashing. In use since 1981 and published around 1984.
 - The [replace](#) program (take a look at it, it's **COOL!**).
 - Connector/ODBC (MyODBC), the ODBC driver for Windows.
 - Fixing bugs in MIT-pthreads to get it to work for MySQL Server. And also Unireg, a curses-based application tool with many utilities.
 - Porting of [mSQL](#) tools like [msqlperl](#), [DBD/DBI](#), and [DB2mysql](#).
 - Most of [crash-me](#) and the foundation for the MySQL benchmarks.
- David Axmark
 - Initial main writer of the **Reference Manual**, including enhancements to [texi2html](#).
 - Automatic Web site updating from the manual.
 - Initial Autoconf, Automake, and Libtool support.
 - Licensing.
 - Parts of all the text files. (Nowadays only the [README](#) is left. The rest ended up in the manual.)
 - Lots of testing of new features.
 - Our in-house Free Software legal expert.
 - Mailing list maintainer (who never has the time to do it right...).
 - Our original portability code (now more than 10 years old). Nowadays only some parts of [mysys](#) are left.
 - Someone for Monty to call in the middle of the night when he just got that new feature to work.
 - Chief "Open Sourcerer" (MySQL community relations).
- Jani Tolonen
 - [mysqlimport](#)
 - A lot of extensions to the command-line clients.

- `PROCEDURE ANALYSE( )`
- Sinisa Milivojevic (now in support)
 - Compression (with `zlib`) in the client/server protocol.
 - Perfect hashing for the lexical analyzer phase.
 - Multi-row `INSERT`
 - `mysqldump -e` option
 - `LOAD DATA LOCAL INFILE`
 - `SQL_CALC_FOUND_ROWS SELECT` option
 - `--max-user-connections=...` option
 - `net_read` and `net_write_timeout`
 - `GRANT/REVOKE` and `SHOW GRANTS FOR`
 - New client/server protocol for 4.0
 - `UNION` in 4.0
 - Multiple-table `DELETE/UPDATE`
 - Subqueries in the `FROM` clause (4.1).
 - User resources management
 - Initial developer of the `MySQL++` C++ API and the `MySQLGUI` client.
- Tonu Samuel (past developer)
 - VIO interface (the foundation for the encrypted client/server protocol).
 - MySQL Filesystem (a way to use MySQL databases as files and directories).
 - The `CASE` expression.
 - The `MD5( )` and `COALESCE( )` functions.
 - `RAID` support for `MyISAM` tables.
- Sasha Pachev (past developer)
 - Initial implementation of replication (up to version 4.0).
 - `SHOW CREATE TABLE`.
 - `mysql-bench`
- Matt Wagner
 - MySQL test suite.
 - Webmaster (until 2002).
- Miguel Solorzano (now in support)
 - Win32 development and release builds.
 - Windows NT server code.

- WinMySQLAdmin
- Timothy Smith (now in development)
 - Dynamic character sets support.
 - configure, RPMs and other parts of the build system.
 - Initial developer of [libmysqld](#), the embedded server.
- Sergei Golubchik
 - Full-text search.
 - Added keys to the [MERGE](#) library.
 - Precision math.
- Jeremy Cole (past developer)
 - Proofreading and editing this fine manual.
 - [ALTER TABLE ... ORDER BY](#)
 - [UPDATE ... ORDER BY](#)
 - [DELETE ... ORDER BY](#)
- Indrek Siitan
 - Designing/programming of our Web interface.
 - Author of our newsletter management system.
- Jorge del Conde (past developer)
 - [MySQLCC \(MySQL Control Center\)](#)
 - Win32 development
 - Initial implementation of the Web site portals.
- Venu Anuganti (past developer)
 - MyODBC 3.51
 - New client/server protocol for 4.1 (for prepared statements).
- Arjen Lentz (also handled community, 2004-2006; now works in Support)
 - Maintainer of the MySQL Reference Manual (2001-2004).
 - Preparing the O'Reilly printed edition of the manual (2002).
- Alexander (Bar) Barkov, Alexey (Holyfoot) Botchkov, and Ramil Kalimullin
 - Spatial data (GIS) and R-Trees implementation for 4.1
 - Unicode and character sets for 4.1; documentation for same
- Oleksandr (Sanja) Byelkin
 - Query cache in 4.0
 - Implementation of subqueries (4.1).

- Implementation of views (5.0).
- Aleksey (Walrus) Kishkin and Alexey (Ranger) Stroganov
 - Benchmarks design and analysis.
 - Maintenance of the MySQL test suite.
- Zak Greant (past employee)
 - Open Source advocate, MySQL community relations.
- Carsten Pedersen
 - The MySQL Certification program.
- Lenz Grimmer
 - Production (build and release) engineering.
- Peter Zaitsev
 - [SHA1\(\)](#), [AES_ENCRYPT\(\)](#) and [AES_DECRYPT\(\)](#) functions.
 - Debugging, cleaning up various features.
- Alexander (Salle) Keremidarski
 - Support.
 - Debugging.
- Per-Erik Martin
 - Lead developer for stored procedures (5.0).
- Jim Winstead
 - Former lead Web developer.
 - Improving server, fixing bugs.
- Mark Matthews
 - Connector/J driver (Java).
- Peter Gultzan
 - SQL standards compliance.
 - Documentation of existing MySQL code/algorithms.
 - Character set documentation.
- Guilhem Bichot
 - Replication, from [MySQL](#) version 4.0.
 - Fixed handling of exponents for [DECIMAL](#).
 - Author of [mysql_tableinfo](#).
 - Backup (in 5.1).
- Antony T. Curtis

- Porting of the MySQL Database software to OS/2.
- Mikael Ronstrom
 - Much of the initial work on NDB Cluster until 2000. Roughly half the code base at that time. Transaction protocol, node recovery, system restart and restart code and parts of the API functionality.
 - Lead Architect, developer, debugger of NDB Cluster 1994-2004
 - Lots of optimizations
- Jonas Orelund
 - On-line Backup
 - The automatic test environment of MySQL Cluster
 - Portability Library for NDB Cluster
 - Lots of other things
- Pekka Nouisiainen
 - Ordered index implementation of MySQL Cluster
 - BLOB support in MySQL Cluster
 - Charset support in MySQL Cluster
- Martin Skold
 - Unique index implementation of MySQL Cluster
 - Integration of NDB Cluster into MySQL
- Magnus Svensson
 - The test framework for MySQL Cluster
 - Integration of NDB Cluster into MySQL
- Tomas Ulin
 - Lots of work on configuration changes for simple installation and use of MySQL Cluster
- Konstantin Osipov
 - Prepared statements.
 - Cursors.
- Dmitri Lenev
 - Time zone support.
 - Triggers (in 5.0).

E.2. Contributors to MySQL

Although MySQL AB owns all copyrights in the [MySQL server](#) and the [MySQL manual](#), we wish to recognize those who have made contributions of one kind or another to the [MySQL distribution](#). Contributors are listed here, in somewhat random order:

- Gianmassimo Vigazzola <qwerg@mbox.vol.it> or <qwerg@tin.it>

The initial port to Win32/NT.

- Per Eric Olsson

For more or less constructive criticism and real testing of the dynamic record format.

- Irena Pancirov <irena@mail.yacc.it>

Win32 port with Borland compiler. [mysqlshutdown.exe](#) and [mysqlwatch.exe](#)

- David J. Hughes

For the effort to make a shareware SQL database. At TcX, the predecessor of MySQL AB, we started with [mSQL](#), but found that it couldn't satisfy our purposes so instead we wrote an SQL interface to our application builder Unireg. [mysqladmin](#) and [mysql client](#) are programs that were largely influenced by their [mSQL](#) counterparts. We have put a lot of effort into making the MySQL syntax a superset of [mSQL](#). Many of the API's ideas are borrowed from [mSQL](#) to make it easy to port free [mSQL](#) programs to the MySQL API. The MySQL software doesn't contain any code from [mSQL](#). Two files in the distribution ([client/inser_t_test.c](#) and [client/select_test.c](#)) are based on the corresponding (non-copyrighted) files in the [mSQL](#) distribution, but are modified as examples showing the changes necessary to convert code from [mSQL](#) to MySQL Server. ([mSQL](#) is copyrighted David J. Hughes.)

- Patrick Lynch

For helping us acquire <http://www.mysql.com/>.

- Fred Lindberg

For setting up gmail to handle the MySQL mailing list and for the incredible help we got in managing the MySQL mailing lists.

- Igor Romanenko <igor@frog.kiev.ua>

[mysqldump](#) (previously [msqldump](#), but ported and enhanced by Monty).

- Yuri Dario

For keeping up and extending the MySQL OS/2 port.

- Tim Bunce

Author of [mysqlhotcopy](#).

- Zarko Mocnik <zarko.mocnik@dem.si>

Sorting for Slovenian language.

- "TAMITO" <tommy@valley.ne.jp>

The [_MB](#) character set macros and the [ujis](#) and [sjis](#) character sets.

- Joshua Chamas <joshua@chamas.com>

Base for concurrent insert, extended date syntax, debugging on NT, and answering on the MySQL mailing list.

- Yves Carlier <Yves.Carlier@rug.ac.be>

[mysqlaccess](#), a program to show the access rights for a user.

- Rhys Jones <rhys@wales.com> (And GWE Technologies Limited)

For one of the early JDBC drivers.

- Dr Xiaokun Kelvin ZHU <X.Zhu@brad.ac.uk>

Further development of one of the early JDBC drivers and other MySQL-related Java tools.

- James Cooper <pixel@organic.com>

For setting up a searchable mailing list archive at his site.

- Rick Mehalick <Rick_Mehalick@i-o.com>

For [xmysql](#), a graphical X client for MySQL Server.

- Doug Sisk <sisk@wix.com>

For providing RPM packages of MySQL for Red Hat Linux.

- Diemand Alexander V. <axeld@vial.ethz.ch>

For providing RPM packages of MySQL for Red Hat Linux-Alpha.

- Antoni Pamies Olive <toni@readysoft.es>

For providing RPM versions of a lot of MySQL clients for Intel and SPARC.

- Jay Bloodworth <jay@pathways.sde.state.sc.us>

For providing RPM versions for MySQL 3.21.

- David Sacerdote <davids@secnet.com>

Ideas for secure checking of DNS hostnames.

- Wei-Jou Chen <jou@nematic.ieo.nctu.edu.tw>

Some support for Chinese(BIG5) characters.

- Wei He <hewei@mail.ied.ac.cn>

A lot of functionality for the Chinese(GBK) character set.

- Jan Pazdziora <adelton@fi.muni.cz>

Czech sorting order.

- Zeev Suraski <bourbon@netvision.net.il>

[FROM_UNIXTIME\(\)](#) time formatting, [ENCRYPT\(\)](#) functions, and [bison](#) advisor. Active mailing list member.

- Luuk de Boer <luuk@wxs.nl>

Ported (and extended) the benchmark suite to [DBI/DBD](#). Have been of great help with [crash-me](#) and running benchmarks. Some new date functions. The [mysql_setpermission](#) script.

- Alexis Mikhailov <root@medinf.chuvashia.su>

User-defined functions (UDFs); [CREATE FUNCTION](#) and [DROP FUNCTION](#).

- Andreas F. Bobak <bobak@relog.ch>

The [AGGREGATE](#) extension to user-defined functions.

- Ross Wakelin <R.Wakelin@march.co.uk>

Help to set up InstallShield for MySQL-Win32.

- Jethro Wright III <jetman@li.net>

The [libmysql.dll](#) library.

- James Pereria <jpereira@iafrica.com>

Mysqlmanager, a Win32 GUI tool for administering MySQL Servers.

- Curt Sampson <cjs@portal.ca>
Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.
- Martin Ramsch <m.ramsch@computer.org>
Examples in the MySQL Tutorial.
- Steve Harvey
For making [mysqlaccess](#) more secure.
- Konark IA-64 Centre of Persistent Systems Private Limited
<http://www.pspl.co.in/konark/>. Help with the Win64 port of the MySQL server.
- Albert Chin-A-Young.
Configure updates for Tru64, large file support and better TCP wrappers support.
- John Birrell
Emulation of [pthread_mutex\(\)](#) for OS/2.
- Benjamin Pflugmann
Extended [MERGE](#) tables to handle [INSERTS](#). Active member on the MySQL mailing lists.
- Jocelyn Fournier
Excellent spotting and reporting innumerable bugs (especially in the MySQL 4.1 subquery code).
- Marc Liyanage
Maintaining the Mac OS X packages and providing invaluable feedback on how to create Mac OS X PKGs.
- Robert Rutherford
Providing invaluable information and feedback about the QNX port.
- Previous developers of NDB Cluster
Lots of people were involved in various ways summer students, master thesis students, employees. In total more than 100 people so too many to mention here. Notable name is Ataullah Dabaghi who up until 1999 contributed around a third of the code base. A special thanks also to developers of the AXE system which provided much of the architectural foundations for NDB Cluster with blocks, signals and crash tracing functionality. Also credit should be given to those who believed in the ideas enough to allocate of their budgets for its development from 1992 to present time.

Other contributors, bugfinders, and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, <jehamby@lightside>, <psmith@BayNetworks.com>, <duane@connect.com.au>, Ted Deppner <ted@psyber.com>, Mike Simons, Jaakko Hyvatti.

And lots of bug report/patches from the folks on the mailing list.

A big tribute goes to those that help us answer questions on the MySQL mailing lists:

- Daniel Koch <dkoch@amcity.com>
Irix setup.
- Luuk de Boer <luuk@wxs.nl>
Benchmark questions.
- Tim Sailer <tps@users.buoy.com>

[DBD::mysql](#) questions.

- Boyd Lynn Gerber <gerberb@zenez.com>

SCO-related questions.

- Richard Mehalick <RM186061@shellus.com>

[xmysql](#)-related questions and basic installation questions.

- Zeev Suraski <bourbon@netvision.net.il>

Apache module configuration questions (log & auth), PHP-related questions, SQL syntax-related questions and other general questions.

- Francesc Guasch <frankie@citel.upc.es>

General questions.

- Jonathan J Smith <jsmith@wtp.net>

Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might need some work.

- David Sklar <sklar@student.net>

Using MySQL from PHP and Perl.

- Alistair MacDonald <A.MacDonald@uel.ac.uk>

Is flexible and can handle Linux and perhaps HP-UX. Tries to get users to use [mysqlbug](#).

- John Lyon <jlyon@imag.net>

Questions about installing MySQL on Linux systems, using either [.rpm](#) files or compiling from source.

- Lorvid Ltd. <lorvid@WOLFENET.com>

Simple billing/license/support/copyright issues.

- Patrick Sherrill <patrick@coconet.com>

ODBC and VisualC++ interface questions.

- Randy Harmon <rjharmon@uptimecomputers.com>

[DBD](#), Linux, some SQL syntax questions.

E.3. Documenters and translators

The following people have helped us with writing the MySQL documentation and translating the documentation or error messages in MySQL.

- Paul DuBois

Ongoing help with making this manual correct and understandable. That includes rewriting Monty's and David's attempts at English into English as other people know it.

- Kim Aldale

Helped to rewrite Monty's and David's early attempts at English into English.

- Michael J. Miller Jr. <mke@terrapin.turbolift.com>

For the first MySQL manual. And a lot of spelling/language fixes for the FAQ (that turned into the MySQL manual a long time ago).

- Yan Cailin

First translator of the MySQL Reference Manual into simplified Chinese in early 2000 on which the Big5 and HK coded (<http://mysql.hitstar.com/>) versions were based. [Personal home page at linuxdb.yeah.net](#).

- Jay Flaherty <fty@mediapulse.com>

Big parts of the Perl [DBI/DBD](#) section in the manual.

- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>

Proof-reading of the Reference Manual.

- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaire.fr>

French error messages.

- Petr Snajdr, <snajdr@pvt.net>

Czech error messages.

- Jaroslaw Lewandowski <jotel@itnet.com.pl>

Polish error messages.

- Miguel Angel Fernandez Roiz

Spanish error messages.

- Roy-Magne Mo <rmo@www.hivolda.no>

Norwegian error messages and testing of MySQL 3.21.xx.

- Timur I. Bakeyev <root@timur.tatarstan.ru>

Russian error messages.

- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>

Italian error messages.

- Dirk Munzinger <dirk@trinity.saar.de>

German error messages.

- Billik Stefan <billik@sun.uniag.sk>

Slovak error messages.

- Stefan Saroiu <tzoompy@cs.washington.edu>

Romanian error messages.

- Peter Feher

Hungarian error messages.

- Roberto M. Serqueira

Portuguese error messages.

- Carsten H. Pedersen

Danish error messages.

- Arjen Lentz

Dutch error messages, completing earlier partial translation (also work on consistency and spelling).

E.4. Libraries used by and included with MySQL

The following is a list of the creators of the libraries we have included with the MySQL server source to make it easy to compile and install MySQL. We are very thankful to all individuals that have created these and it has made our life much easier.

- Fred Fish

For his excellent C debugging and trace library. Monty has made a number of smaller improvements to the library (speed and additional options).

- Richard A. O'Keefe

For his public domain string library.

- Henry Spencer

For his regex library, used in `WHERE column REGEXP regexp`.

- Chris Provenzano

Portable user level pthreads. From the copyright: This product includes software developed by Chris Provenzano, the University of California, Berkeley, and contributors. We are currently using version 1_60_beta6 patched by Monty (see [mit-pthreads/Changes-mysql](#)).

- Jean-loup Gailly and Mark Adler

For the zlib library (used on MySQL on Windows).

- Bjorn Benson

For his safe_malloc (memory checker) package which is used in when you configure MySQL with `--debug`.

- Free Software Foundation

The [readline](#) library (used by the `mysql` command-line client).

- The NetBSD foundation

The [libedit](#) package (optionally used by the `mysql` command-line client).

E.5. Packages that support MySQL

The following is a list of creators/maintainers of some of the most important API/packages/applications that a lot of people use with MySQL.

We can't list every possible package here because the list would then be way too hard to maintain. For other packages, please refer to the software portal at <http://solutions.mysql.com/software/>.

- Tim Bunce, Alligator Descartes

For the [DBD](#) (Perl) interface.

- Andreas Koenig <a.koenig@mind.de>

For the Perl interface for MySQL Server.

- Jochen Wiedmann <wiedmann@neckar-alb.de>

For maintaining the Perl `DBD::mysql` module.

- Eugene Chan <eugene@acenet.com.sg>

For porting PHP for MySQL Server.

- Georg Richter

MySQL 4.1 testing and bug hunting. New PHP 5.0 `mysqli` extension (API) for use with MySQL 4.1 and up.

- Giovanni Maruzzelli <maruzz@matrice.it>

For porting iODBC (Unix ODBC).

- Xavier Leroy <Xavier.Leroy@inria.fr>

The author of LinuxThreads (used by the MySQL Server on Linux).

E.6. Tools that were used to create MySQL

The following is a list of some of the tools we have used to create MySQL. We use this to express our thanks to those that has created them as without these we could not have made MySQL what it is today.

- Free Software Foundation

From whom we got an excellent compiler (`gcc`), an excellent debugger (`gdb`) and the `libc` library (from which we have borrowed `strto.c` to get some code working in Linux).

- Free Software Foundation & The XEmacs development team

For a really great editor/environment used by almost everybody at MySQL AB.

- Julian Seward

Author of `valgrind`, an excellent memory checker tool that has helped us find a lot of otherwise hard to find bugs in MySQL.

- Dorothea Lütkehaus and Andreas Zeller

For `DDD` (The Data Display Debugger) which is an excellent graphical front end to `gdb`.

E.7. Supporters of MySQL

Although MySQL AB owns all copyrights in the `MySQL server` and the `MySQL manual`, we wish to recognize the following companies, which helped us finance the development of the `MySQL server`, such as by paying us for developing a new feature or giving us hardware for development of the `MySQL server`.

- VA Linux / Andover.net

Funded replication.

- NuSphere

Editing of the MySQL manual.

- Stork Design studio

The MySQL Web site in use between 1998-2000.

- Intel

Contributed to development on Windows and Linux platforms.

- Compaq

Contributed to Development on Linux/Alpha.

- SWSoft

Development on the embedded `mysqld` version.

- FutureQuest

`--skip-show-database`

Symbols

! (logical NOT), 624
 != (not equal), 621
 ", 541
 #mysql50 identifier prefix, 541, 544
 %, 646
 % (modulo), 649
 % (wildcard character), 538
 & (bitwise AND), 687
 && (logical AND), 624
 () (parentheses), 618
 (Control-Z) \Z, 538, 751
 * (multiplication), 645
 + (addition), 645
 - (subtraction), 645
 - (unary minus), 645
 --bind-address option (ndb_mgmd), 1059
 --config-file option (ndb_mgmd), 1059
 --connect-string option (MySQL Cluster), 1057
 --debug option (MySQL Cluster), 1057
 --execute option (MySQL Cluster), 1057
 --help option
 MySQL Cluster programs, 1056
 --initial option (ndbd), 1058
 --initial-start option (ndbd), 1058
 --ndb-connectstring option (MySQL Cluster), 1057
 --ndb-log-update-as-write (mysqld option), 1111, 1111
 --ndb-log-updated-only (mysqld option), 1111
 --ndbcluster option (MySQL Cluster), 1057
 --nodaemon option (ndbd), 1059
 --nostart option (ndbd), 1059
 --nowait-nodes option (ndbd), 1058
 --password option, 333
 --usage option
 MySQL Cluster programs, 1056
 --version option (MySQL Cluster), 1057
 -? option
 MySQL Cluster programs, 1056
 -c option (MySQL Cluster), 1057
 -c option (ndb_mgmd) (OBSOLETE), 1059
 -e option (MySQL Cluster), 1057
 -f option (ndb_mgmd), 1059
 -n option (ndbd), 1059
 -p option, 333
 -V option (MySQL Cluster), 1057
 .my.cnf file, 176, 176, 314, 320, 333, 382
 .mysql_history file, 490
 .pid (process ID) file, 355
 / (division), 646
 /etc/passwd, 303, 758
 := (assignment), 550
 < (less than), 621
 <<, 167
 << (left shift), 687
 <= (less than or equal), 621
 <=> (equal to), 620
 <> (not equal), 621
 = (assignment), 550
 = (equal), 620
 > (greater than), 621
 >= (greater than or equal), 621
 >> (right shift), 687
 [API] (MySQL Cluster), 1046
 [MG] (MySQL Cluster), 1045
 [NDBD] (MySQL Cluster), 1039
 [NDBD_DEFAULT] (MySQL Cluster), 1039
 [NDB_MGMD] (MySQL Cluster), 1045
 [SQL] (MySQL Cluster), 1046
 \" (double quote), 537
 \' (single quote), 537
 \. (mysql client command), 163, 494
 \0 (ASCII 0), 537, 751
 \b (backspace), 537, 751
 \n (linefeed), 538, 751
 \n (newline), 538, 751
 \N (NULL), 751
 \r (carriage return), 538, 751
 \t (tab), 538, 751
 \Z (Control-Z) ASCII 26, 538, 751
 \\ (escape), 538
 ^ (bitwise XOR), 687
 _ (wildcard character), 538
 _rowid, 724
 ` , 541
 | (bitwise OR), 686
 || (logical OR), 625
 ~, 687

A

abort-slave-event-count option
 mysqld, 202
 aborted clients, 1775
 aborted connection, 1775
 ABS(), 647
 access control, 314
 access denied errors, 1768
 access privileges, 306
 account privileges
 adding, 328
 accounts
 anonymous user, 96
 root, 96
 ACID, 23, 860
 ACLs, 306
 ACOS(), 647
 Active Server Pages (ASP), 1434
 ActiveState Perl, 132
 add-drop-database option
 mysqldump, 512
 add-drop-table option
 mysqldump, 512
 add-locks option
 mysqldump, 512
 add-user option
 mysqlmanager, 284
 ADDDATE(), 654
 adding
 character sets, 357
 native functions, 1728
 new account privileges, 328
 new functions, 1718
 new user privileges, 328
 new users, 71, 75
 procedures, 1729

- user-defined functions, 1720
- addition (+), 645
- ADDTIME(), 655
- addtodest option
 - mysqlhotcopy, 521
- AES_DECRYPT(), 688
- AES_ENCRYPT(), 688
- After create
 - thread state, 453
- age
 - calculating, 154
- alias, 1788
- alias names
 - case sensitivity, 542
- aliases
 - for expressions, 706
 - for tables, 756
 - in GROUP BY clauses, 706
 - names, 540
 - on expressions, 756
- ALL, 759, 771
- all-databases option
 - mysqlcheck, 509
 - mysqldump, 513
- all-in-1 option
 - mysqlcheck, 509
- all-tablespaces option
 - mysqldump, 513
- allocating local table
 - thread state, 457
- allow-keywords option
 - mysqldump, 513
- allow-suspicious-udfs option
 - mysqld, 202, 303
- allowold option
 - mysqlhotcopy, 521
- ALLOW_INVALID_DATES SQL mode, 270
- ALTER COLUMN, 710
- ALTER DATABASE, 707
- ALTER EVENT, 1217
- ALTER FUNCTION, 1195
- ALTER LOGFILE GROUP, 707
 - (see also MySQL Cluster Disk Data)
- ALTER PROCEDURE, 1195
- ALTER SCHEMA, 707
- ALTER SERVER, 708
- ALTER TABLE, 708, 711, 1791
- ALTER TABLESPACE, 715
 - (see also MySQL Cluster Disk Data)
- ALTER VIEW, 1227
- altering
 - database, 707
 - schema, 707
- analyze option
 - myisamchk, 478
 - mysqlcheck, 509
- ANALYZE TABLE, 800
- Analyzing
 - thread state, 453
- AND
 - bitwise, 687
 - logical, 624
- angel-pid-file option
 - mysqlmanager, 284
- anonymous user, 96, 97, 315, 317

- ANSI mode
 - running, 20
- ansi option
 - mysqld, 202
- ANSI SQL mode, 270, 274
- ANSI_QUOTES SQL mode, 270
- answering questions
 - etiquette, 15
- ANY, 770
- Apache, 171
- API node (MySQL Cluster)
 - defined, 988
- API nodes (see SQL nodes)
- API's
 - list of, 2081
- APIs, 1268
 - Perl, 1358
- apply_status table (OBSOLETE), 1101
 - (see also MySQL Cluster replication)
- approximate-value literals, 1261
- ArbitrationDelay, 1016, 1034
- ArbitrationRank, 1015, 1034
- ArbitrationTimeout (DEPRECATED), 1030
- arbitrator, 1742, 1747
- ARCHIVE storage engine, 849, 922
- Area(), 1182, 1183
- argument processing, 1723
- arithmetic expressions, 645
- arithmetic functions, 686
- AS, 756, 760
- AsBinary(), 1179
- ASCII(), 628
- ASIN(), 647
- AsText(), 1179
- asynchronous replication (see MySQL Cluster replication)
- ATAN(), 647
- ATAN2(), 647
- attackers
 - security against, 302
- auto-generate-sql option
 - mysqslap, 527, 528, 528
- AUTO-INCREMENT
 - ODBC, 1430
- auto-rehash option
 - mysql, 485
- auto-repair option
 - mysqlcheck, 509
- autoclose option
 - mysqld_safe, 276
- AUTO_INCREMENT, 167
 - and NULL values, 1788
- AVG(), 701
- AVG(DISTINCT), 701

B

- backslash
 - escape character, 537
- backspace (\b), 537, 751
- backup option
 - myisamchk, 476
 - myisampack, 480
- BACKUP TABLE, 800
- BackupDataBufferSize, 1078
- BackupDataBufferSize (MySQL Cluster configuration parameter), 1032

- BackupDataDir, 1017
- BackupLogBufferSize, 1032, 1078
- BackupMaxWriteSize, 1033, 1078
- BackupMemory, 1033, 1078
- BackupReportFrequency, 1033
- backups, 341
 - database, 800
 - in MySQL Cluster, 1072, 1072, 1073, 1078
 - in MySQL Cluster replication, 1106
- backups, troubleshooting
 - in MySQL Cluster, 1078
- BackupWriteSize, 1033, 1078
- base64-output option
 - mysqlbinlog, 503
- basedir option
 - mysql.server, 280
 - mysqld, 202
 - mysqld_safe, 276
 - mysql_upgrade, 299
- batch mode, 162
- batch option
 - mysql, 485
- BatchByteSize, 1034
- batched updates (MySQL Cluster Replication), 1104
- BatchSize, 1034
- BatchSizePerLocalScan, 1022
- BdMPolyFromText(), 1175
- BdMPolyFromWKB(), 1176
- BdPolyFromText(), 1175
- BdPolyFromWKB(), 1176
- BEGIN, 782, 1197
 - XA transactions, 789
- benchmark suite, 386
- BENCHMARK(), 692
- benchmarks, 386
- BETWEEN ... AND, 622
- big5, 1749, 1751
- Big5 Chinese character encoding, 1786
- BIGINT data type, 585
- BIN(), 629
- BINARY, 678
- BINARY data type, 589, 601
- binary distributions, 37
 - installing, 70
 - on Linux, 106
- binary log, 368
- binary logging
 - and MySQL Cluster, 1125
- bind-address option
 - mysqld, 202
 - mysqlmanager, 284
- Binlog Dump
 - thread command, 451
- binlog-do-db option
 - mysqld, 369
- binlog-format option
 - mysqld, 202
- binlog-ignore-db option
 - mysqld, 369
- binlog-row-event-max-size option
 - mysqld, 202
- binlog_index table (OBSOLETE), 1100
 - (see also MySQL Cluster replication)
- BIT data type, 584
- BitKeeper tree, 78

- BIT_AND(), 701
- BIT_COUNT, 167
- BIT_COUNT(), 687
- bit_functions
 - example, 167
- BIT_LENGTH(), 629
- BIT_OR, 167
- BIT_OR(), 701
- BIT_XOR(), 701
- BLACKHOLE storage engine, 849, 924
- BLOB
 - inserting binary data, 539
 - size, 608
- BLOB columns
 - default values, 602
 - indexing, 430, 725
- BLOB data type, 589, 602
- block-search option
 - myisamchk, 478
- BOOL data type, 584
- BOOLEAN data type, 584
- bootstrap option
 - mysqld, 202
- Borland Builder 4, 1434
- Boundary(), 1180
- brackets
 - square, 584
- brief option
 - mysqlaccess, 496
- buffer sizes
 - client, 1268
 - mysqld server, 441
- Buffer(), 1185
- bug reports
 - criteria for, 16
- bugs
 - known, 1793
 - reporting, 16
- bugs database, 16
- bugs.mysql.com, 16
- building
 - client programs, 1355

C

- C API
 - data types, 1272
 - functions, 1276
 - linking problems, 1355
- C Prepared statement API
 - functions, 1327
- C++ APIs, 1358
- C++ Builder, 1435
- C++ compiler
 - gcc, 76
- C++ compiler cannot create executables, 81
- C:\my.cnf file, 382
- CACHE INDEX, 832
- caches
 - clearing, 833
- calculating
 - dates, 154
- calendar, 668
- CALL, 1196
- calling sequences for aggregate functions

- UDF, 1722
- calling sequences for simple functions
 - UDF, 1721
- can't create/write to file, 1777
- carriage return (r), 538, 751
- CASE, 625, 1202
- case sensitivity
 - in access checking, 309
 - in identifiers, 542
 - in names, 542
 - in searches, 1786
 - in string comparisons, 637
- case-sensitivity
 - of database names, 21
 - of table names, 21
- CAST, 679
- cast functions, 678
- cast operators, 678
- casts, 618, 620, 678
- CC environment variable, 76, 76, 82, 131
- cc1plus problems, 81
- CEILING(), 648
- Centroid(), 1183
- CFLAGS environment variable, 76, 82, 131
- CHANGE MASTER TO, 839
 - in MySQL Cluster, 1102
- Change user
 - thread command, 451
- ChangeLog, 1837
- changes
 - log, 1837
 - MySQL 5.1, 1837
- changes to privileges, 319
- changing
 - column, 710
 - column order, 1792
 - field, 710
 - table, 708, 711, 1791
- Changing master
 - thread state, 460
- changing socket location, 76, 94, 1785
- CHAR data type, 588, 600
- CHAR VARYING data type, 589
- CHAR(), 629
- CHARACTER data type, 588
- character sets, 77, 355
 - adding, 357
- Character sets, 554
- CHARACTER VARYING data type, 589
- character-set-client-handshake option
 - mysqld, 202
- character-set-filesystem option
 - mysqld, 203
- character-set-server option
 - mysqld, 203
- character-sets-dir option
 - myisamchk, 476
 - myisampack, 481
 - mysql, 486
 - mysqladmin, 501
 - mysqlbinlog, 503
 - mysqlcheck, 509
 - mysqld, 202
 - mysqldump, 513
 - mysqlexport, 523
 - mysqlshow, 525
- characters
 - multi-byte, 359
- CHARACTER_LENGTH(), 629
- CHARACTER_SETS
 - INFORMATION_SCHEMA table, 1240
- charset option
 - comp_err, 295
- CHARSET(), 692
- CHAR_LENGTH(), 629
- check option
 - myisamchk, 475
 - mysqlcheck, 509
- check options
 - myisamchk, 475
- CHECK TABLE, 801
- check-only-changed option
 - myisamchk, 475
 - mysqlcheck, 509
- check-password-file option
 - mysqlmanager, 284
- check-upgrade option
 - mysqlcheck, 509
- checking
 - tables for errors, 347
- Checking master version
 - thread state, 458
- Checking table
 - thread state, 453
- CHECKPOINT Events (MySQL Cluster), 1066
- checkpoint option
 - mysqlhotcopy, 521
- Checksum, 1035
- Checksum (MySQL Cluster), 1037, 1038
- checksum errors, 111
- CHECKSUM TABLE, 803
- Chinese, 1786
- Chinese, Japanese, Korean character sets
 - frequently asked questions, 1749
- choosing
 - a MySQL version, 33
- choosing types, 609
- chroot option
 - mysqld, 203
 - mysqlhotcopy, 521
- circular replication
 - in MySQL Cluster, 1096
- CJK
 - FAQ, 1749
- CJK (Chinese, Japanese, Korean)
 - Access, PHP, etc., 1749, 1753
 - availability of specific characters, 1749, 1756
 - available character sets, 1749, 1756
 - big5, 1749, 1751
 - character sets available, 1749, 1756
 - characters displayed as question marks, 1749
 - CJKV, 1749, 1758
 - collations, 1749, 1749, 1757, 1757
 - conversion problems with Japanese character sets, 1749, 1752
 - data truncation, 1749, 1753
 - Database and table names, 1749, 1758
 - documentation in Chinese, 1749, 1758
 - documentation in Japanese, 1749, 1758
 - documentation in Korean, 1749, 1758
 - gb2312, gbk, 1749, 1751

- Japanese character sets, 1749, 1752
- Korean character set, 1749, 1753
- LIKE and FULLTEXT, 1749, 1755
- MySQL 4.0 behavior, 1749, 1754
- ORDER BY treatment, 1749, 1749, 1757, 1757
- problems with Access, PHP, etc., 1749, 1753
- problems with Big5 character sets (Chinese), 1749, 1751
- problems with data truncation, 1749, 1753
- problems with euckr character set (Korean), 1749, 1753
- problems with GB character sets (Chinese), 1749, 1751
- problems with LIKE and FULLTEXT, 1749, 1755
- problems with Yen sign (Japanese), 1749, 1752
- rejected characters, 1749, 1758
- sort order problems, 1749, 1749, 1757, 1757
- sorting problems, 1749, 1749, 1757, 1757
- testing availability of characters, 1749, 1756
- Unicode collations, 1749, 1757
- Vietnamese, 1749, 1758
- Yen sign, 1749, 1752
- clean-password-file option
 - mysqlmanager, 284
- cleaning up
 - thread state, 453
- clear option
 - mysql_tableinfo, 532
- clear-only option
 - mysql_tableinfo, 532
- Clearing
 - thread state, 461
- clearing
 - caches, 833
- client programs, 468
 - building, 1355
- client tools, 1268
- clients
 - debugging, 140
 - threaded, 1355
- CLOSE, 1201
- Close stmt
 - thread command, 451
- closing
 - tables, 439
- closing tables
 - thread state, 453
- cluster database (OBSOLETE), 1100
 - (see also MySQL Cluster replication)
- cluster logs, 1064, 1065
- cluster_binlog_index table (OBSOLETE) (see MySQL Cluster replication)
- Clustering (see MySQL Cluster)
- CLUSTERLOG commands (MySQL Cluster), 1065
- CLUSTERLOG STATISTICS command (MySQL Cluster), 1069
- cluster_replication database (OBSOLETE) (see MySQL Cluster replication)
- CMake, 85
- COALESCE(), 622
- COERCIBILITY(), 693
- col option
 - mysql_tableinfo, 532
- ColdFusion, 1435
- collating
 - strings, 358
- COLLATION(), 693
- collation-server option
 - mysqld, 203

- COLLATIONS
 - INFORMATION_SCHEMA table, 1240
- COLLATION_CHARACTER_SET_APPLICABILITY
 - INFORMATION_SCHEMA table, 1241
- column
 - changing, 710
 - types, 584
- column comments, 723
- column format, 723
- column names
 - case sensitivity, 542
- column storage, 724
- column-names option
 - mysql, 486
- column-type-info option
 - mysql, 486
- columns
 - changing, 1792
 - indexes, 430
 - names, 540
 - other types, 609
 - selecting, 152
 - storage requirements, 606
- COLUMNS
 - INFORMATION_SCHEMA table, 1237
- columns option
 - mysqlimport, 523
- COLUMN_PRIVILEGES
 - INFORMATION_SCHEMA table, 1239
- comma-separated values data, reading, 750, 759
- command options
 - mysql, 485
 - mysqladmin, 500
 - mysqld, 201
- command options (MySQL Cluster), 1056
 - mysqld, 1057
 - ndbd, 1058
 - ndb_mgm, 1060
 - ndb_mgmd, 1059
- command syntax, 3
- command-line history
 - mysql, 490
- commands
 - for binary distribution, 70
- commands out of sync, 1777
- Comment syntax, 552
- comments
 - adding, 552
 - starting, 26
- comments option
 - mysqldump, 513
- COMMIT, 23, 782
 - XA transactions, 789
- commit option
 - mysqlaccess, 496
- Committing events to binlog
 - thread state, 460
- compact option
 - mysqldump, 513
- comparison operators, 619
- compatibility
 - between MySQL versions, 100
 - with mSQL, 639
 - with ODBC, 542, 586, 619, 621, 723, 762
 - with Oracle, 21, 703, 779

- with PostgreSQL, 22
- with standard SQL, 19
- with Sybase, 782
- compatible option
 - mysqldump, 513
- compiler
 - C++ gcc, 76
- compiling
 - on Windows, 87
 - optimizing, 440
 - problems, 81
 - speed, 461
 - statically, 76
 - user-defined functions, 1726
- complete-insert option
 - mysqldump, 513
- compliance
 - Y2K, 599
- composite partitioning, 1143
- compress option
 - mysql, 486
 - mysqladmin, 501
 - mysqlcheck, 509
 - mysqldump, 513
 - mysqlexport, 523
 - mysqlshow, 525
 - mysqlslap, 527
- COMPRESS(), 689
- compressed tables, 858
- comp_err, 182
 - charset option, 295
 - debug option, 295
 - debug-info option, 295
 - header_file option, 295
 - help option, 295
 - in_file option, 295
 - name_file option, 295
 - out_dir option, 295
 - out_file option, 295
 - statefile option, 295
 - version option, 296
- CONCAT(), 630
- CONCAT_WS(), 630
- concurrency option
 - mysqlslap, 527
- concurrent inserts, 425, 428
- Conditions, 1198
- config-file option
 - mysqld_multi, 281
 - my_print_defaults, 471
 - ndb_config, 1080
- config.cache, 81
- config.cache file, 81
- config.ini (MySQL Cluster), 1003, 1011, 1011, 1056
- configuration
 - MySQL Cluster, 1038
- configuration files, 320
- configuration options, 75
- configure
 - disable-grant-options option, 78
 - enable-thread-safe-client option, 77
 - localstatedir option, 75
 - previx option, 75
 - running after prior invocation, 81
 - with-big-tables option, 78
 - with-charset option, 77
 - with-collation option, 77
 - with-debug option, 77
 - with-embedded-server option, 75
 - with-extra-charsets option, 77, 77
 - with-unix-socket-path option, 76
 - without-server option, 75
- configure option
 - with-low-memory, 81
- configure script, 75
- configuring backups
 - in MySQL Cluster, 1078
- configuring MySQL Cluster, 997, 1008, 1053, 1056
- Configuring MySQL Cluster (concepts), 988
- conflict resolution
 - enabling, 1111
 - in MySQL Cluster Replication, 1110
 - mysqld startup options, 1111
- Connect
 - thread command, 451
- Connect Out
 - thread command, 451
- connecting
 - remotely with SSH, 340
 - to the server, 144, 313
 - verification, 314
- Connecting to master
 - thread state, 458
- connection
 - aborted, 1775
- CONNECTION Events (MySQL Cluster), 1066
- CONNECTION_ID(), 693
- Connector/JDBC, 1361
- Connector/MXJ, 1361
- Connector/NET, 1361, 1440
 - reporting problems, 1605
- Connector/ODBC, 1361, 1361
 - Borland, 1434
 - Borland Database Engine, 1434
 - reporting problems, 1439, 1439
- Connectors
 - MySQL, 1361
- connectstring (see MySQL Cluster)
- connect_timeout variable, 490, 502
- console option
 - mysqld, 203
- constant table, 389, 397
- constraints, 27
- CONSTRAINTS
 - INFORMATION_SCHEMA table, 1241
- Contains(), 1186
- contributing companies
 - list of, 2082
- contributors
 - list of, 2075
- control access, 314
- control flow functions, 625
- CONV(), 630
- conventions
 - typographical, 2
- CONVERT, 679
- CONVERT TO, 712
- converting HEAP to MyISAM
 - thread state, 453
- CONVERT_TZ(), 655

- ConvexHull(), 1185
 - copy option
 - mysqlaccess, 496
 - copy to tmp table
 - thread state, 453
 - copying databases, 103
 - copying tables, 732
 - Copying to group table
 - thread state, 453
 - Copying to tmp table
 - thread state, 453
 - Copying to tmp table on disk
 - thread state, 453
 - core-file option
 - mysqld, 203
 - core-file-size option
 - mysqld_safe, 276
 - correct-checksum option
 - myisamchk, 476
 - correlated subqueries, 773
 - COS(), 648
 - COT(), 648
 - count option
 - myisam_ftdump, 472
 - mysqladmin, 501
 - mysqlshow, 526
 - COUNT(), 701
 - COUNT(DISTINCT), 702
 - counting
 - table rows, 158
 - crash, 136
 - recovery, 347
 - repeated, 1782
 - crash-me, 386
 - crash-me program, 384, 386
 - CRC32(), 648
 - CREATE DATABASE, 716
 - Create DB
 - thread command, 451
 - CREATE EVENT, 1218
 - CREATE FUNCTION, 1192, 1719
 - CREATE INDEX, 716
 - CREATE LOGFILE GROUP, 719
 - (see also MySQL Cluster Disk Data)
 - create option
 - mysqslap, 527
 - CREATE PROCEDURE, 1192
 - CREATE SCHEMA, 716
 - CREATE SERVER, 719
 - CREATE TABLE, 720
 - CREATE TABLESPACE, 734
 - (see also MySQL Cluster Disk Data)
 - CREATE TRIGGER, 1208
 - CREATE USER, 791
 - CREATE VIEW, 1227
 - create-options option
 - mysqldump, 513
 - create-schema option
 - mysqslap, 527
 - creating
 - bug reports, 16
 - database, 716
 - databases, 147
 - default startup options, 176
 - function, 1719
 - schema, 716
 - tables, 148
 - Creating delayed handler
 - thread state, 457
 - Creating index
 - thread state, 453
 - Creating sort index
 - thread state, 453
 - creating table
 - thread state, 454
 - Creating table from master dump
 - thread state, 460
 - Creating tmp table
 - thread state, 454
 - creating user accounts, 791
 - CROSS JOIN, 760
 - Crosses(), 1186
 - CR_SERVER_GONE_ERROR, 1772
 - CR_SERVER_LOST_ERROR, 1772
 - CSV data, reading, 750, 759
 - csv option
 - mysqslap, 527
 - CSV storage engine, 849, 923
 - CURDATE(), 655
 - CURRENT_DATE, 655
 - CURRENT_TIME, 655
 - CURRENT_TIMESTAMP, 656
 - CURRENT_USER(), 693
 - Cursors, 1200
 - CURTIME(), 655
 - customers
 - of MySQL, 385
 - CXX environment variable, 76, 76, 81, 81, 82, 131
 - CXXFLAGS environment variable, 76, 82, 131
- ## D
- Daemon
 - thread command, 451
 - data
 - character sets, 355
 - importing, 494
 - loading into tables, 150
 - retrieving, 150
 - size, 429
 - data node (MySQL Cluster)
 - defined, 988
 - Data truncation with CJK characters, 1749, 1753
 - data type
 - BIGINT, 585
 - BINARY, 589, 601
 - BIT, 584
 - BLOB, 589, 602
 - BOOL, 584, 609
 - BOOLEAN, 584, 609
 - CHAR, 588, 600
 - CHAR VARYING, 589
 - CHARACTER, 588
 - CHARACTER VARYING, 589
 - DATE, 587, 594
 - DATETIME, 587, 594
 - DEC, 587
 - DECIMAL, 586, 1261
 - DOUBLE, 586
 - DOUBLE PRECISION, 586

- ENUM, 590, 603
- FIXED, 587
- FLOAT, 586, 586, 586
- GEOMETRY, 1174
- GEOMETRYCOLLECTION, 1174
- INT, 585
- INTEGER, 585
- LINESTRING, 1174
- LONG, 602
- LOB, 590
- LONGTEXT, 590
- MEDIUMBLOB, 589
- MEDIUMINT, 585
- MEDIUMTEXT, 589
- MULTILINESTRING, 1174
- MULTIPOINT, 1174
- MULTIPOLYGON, 1174
- NATIONAL CHAR, 588
- NCHAR, 588
- NUMERIC, 587
- POINT, 1174
- POLYGON, 1174
- REAL, 586
- SET, 590, 604
- SMALLINT, 585
- TEXT, 589, 602
- TIME, 587, 598
- TIMESTAMP, 587, 594
- TINYBLOB, 589
- TINYINT, 584
- TINYTEXT, 589
- VARBINARY, 589, 601
- VARCHAR, 589, 600
- VARCHARACTER, 589
- YEAR, 587, 599
- data types, 584
 - C API, 1272
 - overview, 584
- data-file-length option
 - myisamchk, 476
- database
 - altering, 707
 - creating, 716
 - deleting, 735
- database design, 429
- Database information
 - obtaining, 810
- database metadata, 1234
- database names
 - case sensitivity, 542
 - case-sensitivity, 21
- database option
 - mysql, 486
 - mysqlbinlog, 503
- DATABASE(), 693
- databases
 - backups, 341
 - copying, 103
 - creating, 147
 - defined, 4
 - information about, 161
 - names, 540
 - replicating, 938
 - selecting, 148
 - symbolic links, 465
 - using, 147
- databases option
 - mysqlcheck, 509
 - mysqldump, 513
- DataDir, 1016, 1017
- datadir option
 - mysql.server, 280
 - mysqld, 203
 - mysqld_safe, 277
 - mysql_upgrade, 300
- DataJunction, 1435
- DataMemory, 1018, 1046
- DATE, 1786
- date and time functions, 652
- Date and Time types, 593
- date calculations, 154
- DATE columns
 - problems, 1786
- DATE data type, 587, 594
- date functions
 - Y2K compliance, 599
- date types, 607
 - Y2K issues, 599
- date values
 - problems, 595
- DATE(), 656
- DATEDIFF(), 656
- dates
 - used with partitioning, 1135
 - used with partitioning (examples), 1137, 1139, 1143, 1158
- DATETIME data type, 587, 594
- DATE_ADD(), 656
- DATE_FORMAT(), 657
- DATE_SUB(), 656, 659
- DAY(), 659
- DAYNAME(), 659
- DAYOFMONTH(), 659
- DAYOFWEEK(), 659
- DAYOFYEAR(), 659
- db option
 - mysqlaccess, 496
- db table
 - sorting, 317
- DB2 SQL mode, 274
- DBI interface, 1358
- DBI->quote, 539
- DBI->trace, 138
- DBI/DBD interface, 1358
- DBI_TRACE environment variable, 131, 138
- DBI_USER environment variable, 131
- DEBUG package, 140
- DEALLOCATE PREPARE, 846, 847
- Debug
 - thread command, 451
- debug option
 - comp_err, 295
 - make_win_bin_dist, 296
 - myisamchk, 474
 - myisampack, 481
 - mysql, 486
 - mysqlaccess, 496
 - mysqladmin, 501
 - mysqlbinlog, 503
 - mysqlcheck, 509
 - mysqld, 203

- mysqldump, 514
- mysqlhotcopy, 521
- mysqlimport, 523
- mysqlmanager, 285
- mysqlshow, 526
- mysqlslap, 527
- my_print_defaults, 471
- debug-check option
 - mysql, 486
 - mysqladmin, 498
 - mysqlbinlog, 503
 - mysqlcheck, 509
 - mysqldump, 514
 - mysqlimport, 523
 - mysqlshow, 526
 - mysqlslap, 527
 - mysql_upgrade, 300
- debug-info option
 - comp_err, 295
 - mysql, 486
 - mysqladmin, 498
 - mysqlbinlog, 503
 - mysqlcheck, 509
 - mysqldump, 514
 - mysqlimport, 523
 - mysqlshow, 526
 - mysqlslap, 527
 - mysql_upgrade, 300
- debugging
 - client, 140
 - server, 136
- debugging support, 75
- DEC data type, 587
- decimal arithmetic, 1261
- DECIMAL data type, 586, 1261
- decimal point, 584
- DECLARE, 1197
- DECODE(), 689
- decode_bits myisamchk variable, 474
- DEFAULT
 - constraint, 28
- default
 - privileges, 96
- default hostname, 313
- default installation location, 44
- default options, 176
- DEFAULT value clause, 590, 723
- default values, 384, 590, 723, 742
 - BLOB and TEXT columns, 602
 - explicit, 590
 - implicit, 590
 - suppression, 28
- DEFAULT(), 698
- default-character-set option
 - mysql, 486
 - mysqladmin, 501
 - mysqlcheck, 509
 - mysqld, 203
 - mysqldump, 514
 - mysqlimport, 523
 - mysqlshow, 526
- default-collation option
 - mysqld, 203
- default-mysqld-path option
 - mysqlmanager, 285
- default-storage-engine option
 - mysqld, 203
- default-table-type option
 - mysqld, 203
- default-time-zone option
 - mysqld, 203
- defaults
 - embedded, 1269
- defaults-extra-file option, 179
 - mysqld_multi, 281
 - mysqld_safe, 277
 - my_print_defaults, 471
- defaults-file option, 179
 - mysqld_multi, 281
 - mysqld_safe, 277
 - mysqlmanager, 285
 - my_print_defaults, 471
- defaults-group-suffix option, 179
 - my_print_defaults, 471
- DEGREES(), 648
- delay-key-write option
 - mysqld, 204, 856
- DELAYED, 745
- Delayed insert
 - thread command, 451
- delayed inserts
 - thread states, 457
- delayed-insert option
 - mysqldump, 514
- delayed_insert_limit, 746
- DELETE, 738
 - and MySQL Cluster, 1122
- delete option
 - mysqlimport, 523
- delete-master-logs option
 - mysqldump, 514
- deleting
 - database, 735
 - foreign key, 711, 881
 - function, 1719
 - index, 710, 735
 - primary key, 711
 - rows, 1789
 - schema, 735
 - table, 736
 - user, 331, 791
 - users, 331, 791
- deleting from main table
 - thread state, 454
- deleting from reference tables
 - thread state, 454
- deletion
 - mysql.sock, 1785
- delimiter option
 - mysql, 486
 - mysqlslap, 528
 - ndb_select_all, 1087
- Delphi, 1434
- derived tables, 773
- des-key-file option
 - mysqld, 204
- DESC, 779
- descending option
 - ndb_select_all, 1087
- DESCRIBE, 161, 779

- description option
 - myisamchk, 478
 - design
 - choices, 429
 - issues, 1793
 - limitations, 384
 - DES_DECRYPT(), 689
 - DES_ENCRYPT(), 689
 - developers
 - list of, 2071
 - development source tree, 78
 - Difference(), 1185
 - digits, 584
 - Dimension(), 1179
 - directory structure
 - default, 44
 - disable-grant-options option
 - configure, 78
 - disable-keys option
 - mysqldump, 514
 - disable-log-bin option
 - mysqlbinlog, 503
 - DISCARD TABLESPACE, 712, 865
 - discard_or_import_tablespace
 - thread state, 454
 - disconnect-slave-event-count option
 - mysqld, 204
 - disconnecting
 - from the server, 144
 - Disjoint(), 1186
 - Disk Data tables (MySQL Cluster) (see MySQL Cluster Disk Data)
 - disk full, 1784
 - disk issues, 464
 - disk option
 - ndb_select_all, 1087
 - DiskCheckpointSpeed, 1028
 - DiskCheckpointSpeedInRestart, 1028
 - Diskless, 1025
 - DiskPageBufferMemory, 1116
 - disks
 - splitting data across, 467
 - DiskSyncSize, 1028
 - display size, 584
 - display triggers, 829
 - display width, 584
 - displaying
 - information
 - Cardinality, 821
 - Collation, 821
 - SHOW, 810, 812, 821, 822, 829
 - table status, 827
 - Distance(), 1186
 - DISTINCT, 153, 415, 759
 - AVG(), 701
 - COUNT(), 702
 - MAX(), 702
 - MIN(), 702
 - SUM(), 703
 - DISTINCTROW, 759
 - DIV, 646
 - division (/), 646
 - DNS, 464
 - DO, 740
 - DocBook XML
 - documentation source format, 1
 - Documentation
 - in Chinese, 1749, 1758
 - in Japanese, 1749, 1758
 - in Korean, 1749, 1758
 - Documenters
 - list of, 2079
 - DOUBLE data type, 586
 - DOUBLE PRECISION data type, 586
 - double quote ("), 537
 - downgrades
 - MySQL Cluster, 1048, 1048, 1050
 - downgrading, 104
 - downloading, 42
 - drbd
 - FAQ, 1759
 - DRBD, 1759, 1759
 - DRBD licence, 1759, 1760
 - DROP DATABASE, 735
 - Drop DB
 - thread command, 451
 - DROP EVENT, 1222
 - DROP FOREIGN KEY, 711, 881
 - DROP FUNCTION, 1196, 1719
 - DROP INDEX, 710, 735
 - DROP LOGFILE GROUP, 736
 - (see also MySQL Cluster Disk Data)
 - DROP PREPARE, 847
 - DROP PRIMARY KEY, 711
 - DROP PROCEDURE, 1196
 - DROP SCHEMA, 735
 - DROP SERVER, 736
 - DROP TABLE, 736
 - and MySQL Cluster, 1122
 - DROP TABLESPACE, 737
 - (see also MySQL Cluster Disk Data)
 - DROP TRIGGER, 1211
 - DROP USER, 791
 - DROP VIEW, 1233
 - drop-user option
 - mysqlmanager, 285
 - dropping
 - user, 331, 791
 - dryrun option
 - mysqlhotcopy, 521
 - DUAL, 756
 - dump option
 - myisam_ftdump, 472
 - DUMPFIL, 759
 - dynamic table characteristics, 858
- ## E
- edit-user option
 - mysqlmanager, 285
 - Eiffel Wrapper, 1359
 - ELT(), 630
 - email lists, 13
 - embedded MySQL server library, 1268
 - embedded option
 - make_win_bin_dist, 296
 - enable-named-pipe option
 - mysqld, 204
 - enable-pstack option
 - mysqld, 205
 - enable-thread-safe-client option

- configure, 77
- ENCODE(), 689
- ENCRYPT(), 690
- encryption, 334
- encryption functions, 687
- end
 - thread state, 454
- END, 1197
- EndPoint(), 1181
- engine option
 - mysqslap, 528
- ENGINES
 - INFORMATION_SCHEMA table, 1246
- ENTER SINGLE USER MODE command (MySQL Cluster), 1063
- entering
 - queries, 145
- ENUM
 - size, 609
- ENUM data type, 590, 603
- Envelope(), 1180
- environment variable
 - CC, 76, 76, 82
 - CFLAGS, 76, 82
 - CXX, 76, 76, 82
 - CXXFLAGS, 76, 82
 - HOME, 490
 - LD_RUN_PATH, 107, 113
 - MYSQL_DEBUG, 470
 - MYSQL_HISTFILE, 490
 - MYSQL_HOST, 314
 - MYSQL_PWD, 314, 470
 - MYSQL_TCP_PORT, 382, 382, 470
 - MYSQL_UNIX_PORT, 382, 382, 470
 - PATH, 71, 173
 - USER, 314
- Environment variable
 - CC, 131
 - CFLAGS, 131
 - CXX, 81, 131
 - CXXFLAGS, 131
 - DBI_TRACE, 131, 138
 - DBI_USER, 131
 - HOME, 131
 - LD_LIBRARY_PATH, 133
 - LD_RUN_PATH, 131, 133
 - MYSQL_DEBUG, 131, 140
 - MYSQL_GROUP_SUFFIX, 131
 - MYSQL_HISTFILE, 131
 - MYSQL_HOME, 131
 - MYSQL_HOST, 131
 - MYSQL_PS1, 131
 - MYSQL_PWD, 131
 - MYSQL_TCP_PORT, 131
 - MYSQL_UNIX_PORT, 92, 131
 - PATH, 131
 - TMPDIR, 92, 131
 - TZ, 131, 1786
 - UMASK, 131, 1780
 - UMASK_DIR, 131, 1780
 - USER, 131
- Environment variables
 - CXX, 81
- environment variables, 180, 320, 470
 - list of, 131
- equal (=), 620
- Equals(), 1186
- Error
 - thread command, 451
- ERROR Events (MySQL Cluster), 1068
- error logs (MySQL Cluster), 1054
- error messages
 - can't find file, 1780
 - languages, 356, 356
- errors
 - access denied, 1768
 - checking tables for, 347
 - common, 1767
 - directory checksum, 111
 - handling for UDFs, 1725
 - in subqueries, 774
 - known, 1793
 - linking, 1779
 - list of, 1768
 - reporting, 1, 16, 16
- ERROR_FOR_DIVISION_BY_ZERO SQL mode, 270
- escape (\), 538
- escape characters, 537
- estimating
 - query performance, 396
- event log format (MySQL Cluster), 1066
- event logs (MySQL Cluster), 1064, 1065, 1065
- event scheduler
 - thread states, 461
- Event Scheduler, 1214
 - altering events, 1217
 - and MySQL privileges, 1223
 - and mysqladmin debug, 1222
 - and SHOW PROCESSLIST, 1215
 - concepts, 1214
 - creating events, 1218
 - dropping events, 1222
 - enabling and disabling, 1215
 - event metadata, 1222
 - obtaining status information, 826, 1222
 - SQL statements, 1216
 - starting and stopping, 1215
- event severity levels (MySQL Cluster), 1065
- event types (MySQL Cluster), 1064, 1066
- event-scheduler option
 - mysqld, 204
- events, 1214
 - altering, 1217
 - creating, 1218
 - dropping, 1222
 - limitations, 1225
 - metadata, 1222
 - status variables, 1225
- EVENTS
 - INFORMATION_SCHEMA table, 1224, 1249
- events option
 - mysqldump, 514
- exact-value literals, 1261
- example option
 - mysqld_multi, 281
- EXAMPLE storage engine, 849, 916
- examples
 - compressed tables, 481
 - myisamchk output, 350
 - queries, 163
- exe-suffix option

- make_win_bin_dist, 296
- Execute
 - thread command, 451
- EXECUTE, 846, 847
- execute option
 - mysql, 486
- ExecuteOnComputer, 1014, 1016, 1034
- executing SQL statements from text files, 162, 494
- Execution of init_command
 - thread state, 454
- EXISTS
 - with subqueries, 772
- EXIT command (MySQL Cluster), 1063
- EXIT SINGLE USER MODE command (MySQL Cluster), 1063
- exit-info option
 - mysqld, 204
- EXP(), 648
- EXPLAIN, 387
- EXPLAIN PARTITIONS, 1155, 1156
- EXPLAIN used with partitioned tables, 1155
- explicit default values, 590
- EXPORT_SET(), 630
- expression aliases, 706, 756
- expressions
 - extended, 156
- extend-check option
 - myisamchk, 476, 476
- extended option
 - mysqlcheck, 510
- extended-insert option
 - mysqldump, 514
- extensions
 - to standard SQL, 19
- ExteriorRing(), 1183
- external locking, 204, 209, 241, 347, 428, 456
- external-locking option
 - mysqld, 204
- extra-file option
 - my_print_defaults, 471
- extra-partition-info option
 - ndb_desc, 1083
- EXTRACT(), 659
- extracting
 - dates, 154
- ExtractValue(), 681

F

- failover
 - in MySQL Cluster replication, 1105
- FALSE, 539, 540
 - testing for, 621
- fast option
 - myisamchk, 476
 - mysqlcheck, 510
- fatal signal 11, 81
- features of MySQL, 5
- FEDERATED storage engine, 849, 917
- Fetch
 - thread command, 451
- FETCH, 1201
- field
 - changing, 710
- Field List
 - thread command, 452

- FIELD(), 631
- fields option
 - ndb_config, 1081
- fields-enclosed-by option
 - mysqldump, 514, 523
- fields-escaped-by option
 - mysqldump, 514, 523
- fields-optionally-enclosed-by option
 - mysqldump, 514, 523
- fields-terminated-by option
 - mysqldump, 514, 523
- FILE, 632
- files
 - binary log, 368
 - config.cache, 81
 - error messages, 356
 - general query log, 367
 - log, 75, 376
 - my.cnf, 967
 - not found message, 1780
 - permissions, 1780
 - repairing, 476
 - script, 162
 - size limits, 1776
 - slow query log, 375
 - text, 494
 - tmp, 92
- FILES
 - INFORMATION_SCHEMA table, 1252
 - (see also MySQL Cluster Disk Data)
- filesort optimization, 412
- FileSystemPath, 1017
- FIND_IN_SET(), 631
- Finished reading one binlog; switching to next binlog
 - thread state, 458
- first-slave option
 - mysqldump, 514
- fix-db-names option
 - mysqlcheck, 510
- fix-table-names option
 - mysqlcheck, 510
- FIXED data type, 587
- fixed-point arithmetic, 1261
- FLOAT data type, 586, 586, 586
- floating-point number, 586
- floats, 539
- FLOOR(), 648
- FLUSH, 833
- flush option
 - mysqld, 205
- flush tables, 500
- flush-logs option
 - mysqldump, 514
- flush-privileges option
 - mysqldump, 514
- Flushing tables
 - thread state, 454
- flushlog option
 - mysqlhotcopy, 521
- FOR UPDATE, 759
- FORCE INDEX, 766, 1791
- FORCE KEY, 766
- force option
 - myisamchk, 476, 477
 - myisampack, 481

- mysql, 486
- mysqladmin, 501
- mysqlcheck, 510
- mysqldump, 515
- mysqlimport, 523
- mysql_upgrade, 300
- force-read option
 - mysqlbinlog, 503
- foreign key
 - constraint, 28
 - deleting, 711, 881
- foreign keys, 25, 165, 711
- FORMAT(), 631
- Forums, 15
- FOUND_ROWS(), 694
- FragmentLogFileSize, 1023
- FreeBSD troubleshooting, 82
- freeing items
 - thread state, 454
- frequently-asked questions about DRBD, 1759
- frequently-asked questions about MySQL Cluster, 1741
- FROM, 756
- FROM_DAYS(), 659
- FROM_UNIXTIME(), 660
- ft_max_word_len myisamchk variable, 474
- ft_min_word_len myisamchk variable, 474
- ft_stopword_file myisamchk variable, 474
- full disk, 1784
- full-text search, 668
- FULLTEXT, 668
- fulltext
 - stopword list, 677
- FULLTEXT initialization
 - thread state, 454
- function
 - creating, 1719
 - deleting, 1719
- function names
 - parsing, 544
 - resolving ambiguity, 544
- functions, 611
 - arithmetic, 686
 - bit, 686
 - C API, 1276
 - C Prepared statement API, 1327
 - cast, 678
 - control flow, 625
 - date and time, 652
 - encryption, 687
 - GROUP BY, 700
 - grouping, 618
 - information, 692
 - mathematical, 647
 - miscellaneous, 697
 - native
 - adding, 1728
 - new, 1718
 - string, 627
 - string comparison, 637
 - user-defined, 1718
 - adding, 1720
- Functions
 - user-defined, 1719, 1719
- functions for SELECT and WHERE clauses, 611
- Future development of MySQL Cluster, 1128

G

- gap lock, 869, 888, 890
- gb2312, gbk, 1749, 1751
- gcc, 76
- gci option
 - ndb_select_all, 1087
- gdb
 - using, 137
- gdb option
 - mysqld, 205
- general information, 1
- General Public License, 4
- general query log, 367
- general-log option
 - mysqld, 205
- geographic feature, 1166
- GeomCollFromText(), 1174
- GeomCollFromWKB(), 1175
- geometry, 1166
- GEOMETRY data type, 1174
- GEOMETRYCOLLECTION data type, 1174
- GeometryCollection(), 1176
- GeometryCollectionFromText(), 1174
- GeometryCollectionFromWKB(), 1175
- GeometryFromText(), 1174
- GeometryFromWKB(), 1175
- GeometryN(), 1184
- GeometryType(), 1180
- GeomFromText(), 1174, 1179
- GeomFromWKB(), 1175, 1179
- geospatial feature, 1166
- getting MySQL, 42
- GET_FORMAT(), 660
- GET_LOCK(), 698
- GIS, 1166, 1166
- GLength(), 1181, 1182
- global privileges, 792, 799
- GLOBAL_STATUS
 - INFORMATION_SCHEMA table, 1258
- GLOBAL_VARIABLES
 - INFORMATION_SCHEMA table, 1258
- goals of MySQL, 5
- got handler lock
 - thread state, 457
- got old table
 - thread state, 457
- GRANT, 792
- GRANT statement, 328
- grant tables, 319
 - re-creating, 92
 - sorting, 316, 317
- granting
 - privileges, 792
- GRANTS, 820
- greater than (>), 621
- greater than or equal (>=), 621
- GREATEST(), 622
- GROUP BY, 413
 - aliases in, 706
 - extensions to standard SQL, 705, 757
- GROUP BY functions, 700
- grouping
 - expressions, 618

GROUP_CONCAT(), 702

H

HANDLER, 740

Handlers, 1199

handling

errors, 1725

Has read all relay log; waiting for the slave I/O thread to update it
thread state, 459

Has sent all binlog to slave; waiting for binlog to be updated
thread state, 458

hash partitioning, 1139

hash partitions

managing, 1153

splitting and merging, 1153

HAVING, 757

header option

ndb_select_all, 1087

header_file option

comp_err, 295

HEAP storage engine, 849, 915

HeartbeatIntervalDbApi, 1027

HeartbeatIntervalDbDb, 1026

HELP command (MySQL Cluster), 1062

help option

comp_err, 295

myisamchk, 474

myisampack, 480

myisam_ftdump, 472

mysql, 485

mysqlaccess, 496

mysqladmin, 501

mysqlbinlog, 503

mysqlcheck, 508

mysqld, 202

mysqldump, 512

mysqld_multi, 281

mysqld_safe, 276

mysqlhotcopy, 521

mysqlimport, 523

mysqlmanager, 284

mysqlshow, 525

mysqlslap, 527

mysql_explain_log, 529

mysql_find_rows, 530

mysql_setpermission, 531

mysql_tableinfo, 532

mysql_upgrade, 299

mysql_waitpid, 533

my_print_defaults, 471

pererror, 534

resolveip, 536

resolve_stack_dump, 536

HELP statement, 780

HEX(), 631

hex-blob option

mysqldump, 515

hexadecimal values, 539

hexdump option

mysqlbinlog, 503

HIGH_NOT_PRECEDENCE SQL mode, 270

HIGH_PRIORITY, 760

hints, 20

index, 756, 766

history of MySQL, 5

HOME environment variable, 131, 490

host option

mysql, 486

mysqlaccess, 496

mysqladmin, 501

mysqlbinlog, 503

mysqlcheck, 510

mysqldump, 515

mysqlhotcopy, 521

mysqlimport, 523

mysqlshow, 526

mysqlslap, 528

mysql_explain_log, 529

mysql_setpermission, 531

mysql_tableinfo, 532

ndb_config, 1081

host table, 318

sorting, 317

Host*SciId* parameters, 1037

host.frm

problems finding, 89

hostname

default, 313

HostName, 1015, 1016, 1034

hostname caching, 464

HOURL(), 661

howto option

mysqlaccess, 497

html option

mysql, 486

I

i-am-a-dummy option

mysql, 488

icc

and MySQL Cluster support>, 135

Id, 1014, 1016, 1034

ID

unique, 1354

id option

ndb_config, 1081

identifiers, 540

case sensitivity, 542

quoting, 541

idx option

mysql_tableinfo, 533

IF, 1202

IF(), 626

IFNULL(), 626

IGNORE INDEX, 766

IGNORE KEY, 766

ignore option

mysqlimport, 523

ignore-lines option

mysqlimport, 523

ignore-spaces option

mysql, 486

ignore-table option

mysqldump, 515

IGNORE_SPACE SQL mode, 271

implicit default values, 590

IMPORT TABLESPACE, 712, 865

importing

- data, 494
 - IN, 622, 770
 - increasing
 - performance, 976, 977
 - increasing with replication
 - speed, 938
 - index
 - deleting, 710, 735
 - index hints, 756, 766
 - indexes, 716
 - and BLOB columns, 430, 725
 - and IS NULL, 433
 - and LIKE, 433
 - and NULL values, 725
 - and TEXT columns, 430, 725
 - assigning to key cache, 832
 - block size, 230
 - columns, 430
 - leftmost prefix of, 432
 - multi-column, 431
 - multiple-part, 716
 - names, 540
 - use of, 431
 - IndexMemory, 1018, 1046
 - INET_ATON(), 698
 - INET_NTOA(), 698
 - INFO Events (MySQL Cluster), 1068
 - information functions, 692
 - information option
 - myisamchk, 476
 - INFORMATION_SCHEMA, 1234
 - INFORMATION_SCHEMA.ENGINES table
 - used with MySQL Cluster, 1071
 - INFORMATION_SCHEMA.GLOBAL_STATUS table
 - used with MySQL Cluster, 1072
 - INFORMATION_SCHEMA.GLOBAL_VARIABLES table
 - used with MySQL Cluster, 1071
 - init
 - thread state, 454
 - Init DB
 - thread command, 452
 - init-file option
 - mysqld, 205
 - Initialized
 - thread state, 461
 - InitialNoOfOpenFiles, 1023
 - INNER JOIN, 760
 - innochecksum, 468
 - InnoDB, 860
 - NFS, 861, 908
 - Solaris 10 x86_64 issues, 111
 - innodb option
 - mysqld, 866
 - InnoDB storage engine, 849, 860
 - InnoDB tables, 23
 - innodb_status_file option
 - mysqld, 866
 - INSERT, 420, 741
 - insert
 - thread state, 457
 - INSERT ... SELECT, 744
 - INSERT DELAYED, 745, 745
 - INSERT statement
 - grant privileges, 329
 - INSERT(), 631
 - insert-ignore option
 - mysqldump, 515
 - inserting
 - speed of, 420
 - inserts
 - concurrent, 425, 428
 - install option
 - mysqlmanager, 285
 - INSTALL PLUGIN, 1705
 - installation layouts, 44
 - installation overview, 72
 - installing
 - binary distribution, 70
 - Linux RPM packages, 63
 - Mac OS X PKG packages, 65
 - overview, 31
 - Perl, 131
 - Perl on Windows, 132
 - Solaris PKG packages, 67
 - source distribution, 72
 - user-defined functions, 1726
 - installing MySQL Cluster, 997, 1000, 1009
 - installing plugins, 1705
 - INSTR(), 632
 - INT data type, 585
 - integer arithmetic, 1261
 - INTEGER data type, 585
 - integers, 539
 - InteriorRingN(), 1183
 - internal compiler errors, 81
 - internal locking, 425
 - internals, 1702
 - Internet Relay Chat, 15
 - Intersection(), 1185
 - Intersects(), 1187
 - INTERVAL(), 623
 - introducer
 - string literal, 537, 558
 - invalid data
 - constraint, 28
 - in_file option
 - comp_err, 295
 - IRC, 15
 - IS boolean_value, 621
 - IS NOT boolean_value, 621
 - IS NOT NULL, 621
 - IS NULL, 403, 621
 - and indexes, 433
 - isamlog, 468
 - IsClosed(), 1182
 - IsEmpty(), 1180
 - ISNULL(), 623
 - ISOLATION LEVEL, 787
 - IsRing(), 1182
 - IsSimple(), 1180
 - IS_FREE_LOCK(), 699
 - IS_USED_LOCK(), 699
 - ITERATE, 1202
 - iterations option
 - mysqlslap, 528
- ## J
- Japanese character sets
 - conversion, 1749, 1752

- Japanese, Korean, Chinese character sets
 - frequently asked questions, 1749
- JOIN, 760
- join option
 - myisampack, 481

K

- keepold option
 - mysqlhotcopy, 521
- Key cache
 - MyISAM, 434
- key cache
 - assigning indexes to, 832
- key partitioning, 1141
- key partitions
 - managing, 1153
 - splitting and merging, 1153
- key space
 - MyISAM, 857
- keys, 430
 - foreign, 25, 165
 - multi-column, 431
 - searching on two, 167
- keys option
 - mysqlshow, 526
- keys-used option
 - myisamchk, 477
- keywords, 547
- key_buffer_size myisamchk variable, 474
- KEY_COLUMN_USAGE
 - INFORMATION_SCHEMA table, 1241
- Kill
 - thread command, 452
- KILL, 835
- Killed
 - thread state, 454
- Killing slave
 - thread state, 460
- known errors, 1793
- Korean, 1749, 1753
- Korean, Chinese, Japanese character sets
 - frequently asked questions, 1749

L

- language option
 - mysqld, 205
- language support
 - error messages, 356
- large-pages option
 - mysqld, 205
- last row
 - unique ID, 1354
- LAST_DAY(), 661
- LAST_INSERT_ID(), 25, 744
- LAST_INSERT_ID() and stored routines, 1204
- LAST_INSERT_ID() and triggers, 1204
- LAST_INSERT_ID([expr]), 694
- layout of installation, 44
- LCASE(), 632
- LD_LIBRARY_PATH environment variable, 133
- LD_RUN_PATH environment variable, 107, 113, 131, 133
- LEAST(), 623
- LEAVE, 1202
- ledir option

- mysqld_safe, 277
- LEFT JOIN, 404, 760
- LEFT OUTER JOIN, 760
- LEFT(), 632
- leftmost prefix of indexes, 432
- legal names, 540
- length option
 - myisam_ftdump, 472
- LENGTH(), 632
- less than (<), 621
- less than or equal (<=), 621
- libmysqld, 1268
 - options, 1269
- libraries
 - list of, 2081
- library
 - mysqlclient, 1268
 - mysqld, 1268
- LIKE, 637
 - and indexes, 433
 - and wildcards, 433
- LIMIT, 419, 694, 758
- limitations
 - design, 384
 - MySQL Limitations, 2070
 - replication, 967
- limitations of MySQL Cluster, 1121
- limits
 - file-size, 1776
 - MySQL Limits, limits in MySQL, 2070
- line-numbers option
 - mysql, 487
- linear hash partitioning, 1140
- linear key partitioning, 1142
- linefeed (\n), 538, 751
- LineFromText(), 1175
- LineFromWKB(), 1175
- lines-terminated-by option
 - mysqldump, 515, 523
- LINESTRING data type, 1174
- LineString(), 1176
- LineStringFromText(), 1175
- LineStringFromWKB(), 1175
- linking, 1355
 - errors, 1779
 - problems, 1355
 - speed, 461
- links
 - symbolic, 465
- Linux
 - binary distribution, 106
 - source distribution, 106
- list partitioning, 1138
- list partitions
 - adding and dropping, 1149
 - managing, 1149
- list-users option
 - mysqlmanager, 285
- literals, 537
- LN(), 649
- LOAD DATA FROM MASTER, 840
- LOAD DATA INFILE, 747, 1788
- LOAD INDEX INTO CACHE
 - and partitioning, 1160
- LOAD TABLE FROM MASTER, 841

- loading
 - tables, 150
 - LOAD_FILE(), 632
 - local checkpoints (MySQL Cluster), 1046
 - local option
 - mysqlexport, 524
 - local-infile option
 - mysql, 487
 - mysqld, 303
 - local-load option
 - mysqlbinlog, 504
 - localstatedir option
 - configure, 75
 - LOCALTIME, 661
 - LOCALTIMESTAMP, 661
 - LOCATE(), 632
 - LOCK IN SHARE MODE, 759
 - lock option
 - ndb_select_all, 1086
 - LOCK TABLES, 785
 - lock-all-tables option
 - mysqldump, 515
 - lock-directory option
 - mysqslap, 528, 529
 - lock-tables option
 - mysqldump, 515
 - mysqlexport, 524
 - Locked
 - thread state, 454
 - locking, 440
 - external, 204, 209, 241, 347, 428, 456
 - internal, 425
 - row-level, 25, 425
 - table-level, 425
 - locking methods, 425
 - LockPagesInMainMemory, 1024
 - log
 - changes, 1837
 - log files, 75, 364
 - maintaining, 376
 - names, 341
 - log files (MySQL Cluster), 1054
 - log option
 - mysqld, 205
 - mysqld_multi, 281
 - mysqlmanager, 285
 - LOG(), 649
 - log-bin option
 - mysqld, 205
 - log-bin-index option
 - mysqld, 206
 - log-bin-trust-function-creators option
 - mysqld, 206
 - log-error option
 - mysqld, 206
 - mysqld_safe, 277
 - log-isam option
 - mysqld, 206
 - log-long-format option
 - mysqld, 206
 - log-output option
 - mysqld, 206
 - log-queries-not-using-indexes option
 - mysqld, 206
 - log-short-format option
 - mysqld, 206
 - log-slave-updates option
 - mysqld, 950
 - log-slow-admin-statements option
 - mysqld, 206
 - log-slow-queries option
 - mysqld, 206
 - log-tc option
 - mysqld, 207
 - log-tc-size option
 - mysqld, 207
 - log-warnings option
 - mysqld, 207, 951
 - LOG10(), 649
 - LOG2(), 649
 - LogDestination, 1015
 - logging commands (MySQL Cluster), 1065
 - logging slow query
 - thread state, 454
 - logical operators, 624
 - login
 - thread state, 454
 - LogLevelCheckpoint (MySQL Cluster configuration parameter), 1031
 - LogLevelCongestion, 1032
 - LogLevelConnection (MySQL Cluster configuration parameter), 1031
 - LogLevelError, 1032
 - LogLevelInfo, 1032
 - LogLevelNodeRestart (MySQL Cluster configuration parameter), 1031
 - LogLevelShutdown, 1031
 - LogLevelStartup, 1031
 - LogLevelStatistic (MySQL Cluster configuration parameter), 1031
 - Long Data
 - thread command, 452
 - LONG data type, 602
 - LOB data type, 590
 - LongMessageBuffer, 1022
 - LONGTEXT data type, 590
 - LOOP, 1202
 - loops option
 - ndb_show_tables, 1088
 - low-priority option
 - mysqlexport, 524
 - low-priority-updates option
 - mysqld, 207
 - LOWER(), 632
 - LPAD(), 633
 - LTRIM(), 633
- ## M
- Mac OS X, 1361
 - installation, 65
 - mailing list address, 1
 - mailing lists, 13
 - archive location, 13
 - guidelines, 15
 - main features of MySQL, 5
 - maintaining
 - log files, 376
 - tables, 355
 - MAKEDATE(), 661
 - MAKETIME(), 661
 - make_binary_distribution, 183
 - MAKE_SET(), 633
 - make_win_bin_dist, 183

- debug option, 296
- embedded option, 296
- exe-suffix option, 296
- no-debug option, 296
- no-embedded option, 296
- only-debug option, 296
- Making temp file
 - thread state, 459
- management node (MySQL Cluster)
 - defined, 988
- managing MySQL Cluster, 1060
- managing MySQL Cluster processes, 1053
- manual
 - available formats, 1
 - online location, 1
 - typographical conventions, 2
- master-connect-retry option
 - mysqld, 951
- master-data option
 - mysqldump, 515
- master-host option
 - mysqld, 951
- master-info-file option
 - mysqld, 951
- master-password option
 - mysqld, 951
- master-port option
 - mysqld, 951
- master-retry-count option
 - mysqld, 951
- master-ssl option
 - mysqld, 951
- master-ssl-ca option
 - mysqld, 951
- master-ssl-capath option
 - mysqld, 951
- master-ssl-cert option
 - mysqld, 951
- master-ssl-cipher option
 - mysqld, 951
- master-ssl-key option
 - mysqld, 951
- master-user option
 - mysqld, 951
- MASTER_POS_WAIT(), 699, 841
- MATCH ... AGAINST(), 668
- matching
 - patterns, 156
- math, 1261
- mathematical functions, 647
- MAX(), 702
- MAX(DISTINCT), 702
- max-binlog-dump-events option
 - mysqld, 207
- max-record-length option
 - myisamchk, 477
- max-relay-log-size option
 - mysqld, 951
- MAXDB SQL mode, 274
- maximum memory used, 500
- maximums
 - maximum tables per join, 2070
- MaxNoOfAttributes, 1023
- MaxNoOfConcurrentIndexOperations, 1021
- MaxNoOfConcurrentOperations, 1020
- MaxNoOfConcurrentScans, 1021
- MaxNoOfConcurrentTransactions, 1020
- MaxNoOfFiredTriggers, 1021
- MaxNoOfIndexes, 1024
- MaxNoOfLocalOperations, 1021
- MaxNoOfLocalScans, 1022
- MaxNoOfOpenFiles, 1023
- MaxNoOfOrderedIndexes, 1024
- MaxNoOfSavedMessages, 1023
- MaxNoOfTables, 1024
- MaxNoOfTriggers, 1024
- MaxNoOfUniqueHashIndexes, 1024
- MaxScanBatchSize, 1035
- max_allowed_packet variable, 490
- MAX_CONNECTIONS_PER_HOUR, 331
- max_join_size variable, 490
- MAX_QUERIES_PER_HOUR, 331
- MAX_UPDATES_PER_HOUR, 331
- MAX_USER_CONNECTIONS, 331
- MBR, 1185
- MBRContains(), 1185
- MBRDisjoint(), 1185
- MBREqual(), 1185
- MBRIntersects(), 1185
- MBROverlaps(), 1185
- MBRTouches(), 1186
- MBRWithin(), 1186
- MD5(), 690
- medium-check option
 - myisamchk, 476
 - mysqlcheck, 510
- MEDIUMBLOB data type, 589
- MEDIUMINT data type, 585
- MEDIUMTEXT data type, 589
- memlock option
 - mysqld, 207
- MEMORY storage engine, 849, 915
- memory usage
 - myisamchk, 478
- memory use, 462, 500
 - in MySQL Cluster, 1122
- MemReportFrequency, 1032
- MERGE storage engine, 849, 911
- MERGE tables
 - defined, 911
- metadata
 - database, 1234
- method option
 - mysqlhotcopy, 521
- methods
 - locking, 425
- mgmd (MySQL Cluster)
 - defined, 988
 - (see also management node (MySQL Cluster))
- MICROSECOND(), 661
- Microsoft Access, 1431
- Microsoft ADO, 1433
- Microsoft Excel, 1433
- Microsoft Visual Basic, 1433
- Microsoft Visual InterDev, 1433
- MID(), 633
- MIN(), 702
- MIN(DISTINCT), 702
- Minimum Bounding Rectangle, 1185
- minus

- unary (-), 645
- MINUTE(), 661
- mirror sites, 42
- miscellaneous functions, 697
- MIT-pthreads, 83
- MLineFromText(), 1175
- MLineFromWKB(), 1175
- MOD (modulo), 649
- MOD(), 649
- modes
 - batch, 162
- modulo (%), 649
- modulo (MOD), 649
- monitor
 - terminal, 144
- monitoring-interval option
 - mysqlmanager, 285
- Mono, 1440
- MONTH(), 662
- MONTHNAME(), 662
- MPointFromText(), 1175
- MPointFromWKB(), 1175
- MPolyFromText(), 1175
- MPolyFromWKB(), 1176
- mSQL compatibility, 639
- MSSQL SQL mode, 274
- multi-byte character sets, 1778
 - and MySQL Cluster (replication), 1096
- multi-byte characters, 359
- multi-column indexes, 431
- MULTILINESTRING data type, 1174
- MultiLineString(), 1176
- MultiLineStringFromText(), 1175
- MultiLineStringFromWKB(), 1175
- multiple servers, 377
- multiple-part index, 716
- multiplication (*), 645
- MULTIPOINT data type, 1174
- MultiPoint(), 1176
- MultiPointFromText(), 1175
- MultiPointFromWKB(), 1175
- MULTIPOLYGON data type, 1174
- MultiPolygon(), 1176
- MultiPolygonFromText(), 1175
- MultiPolygonFromWKB(), 1176
- My
 - derivation, 5
- my.cnf
 - and MySQL Cluster, 1003, 1011, 1011
 - in MySQL Cluster, 1053
- my.cnf file, 967
- MyISAM
 - compressed tables, 858
 - size, 606
- MyISAM key cache, 434
- MyISAM storage engine, 849, 854
- mysam-recover option
 - mysqld, 207, 856
- mysamchk, 77, 468
 - analyze option, 478
 - backup option, 476
 - block-search option, 478
 - character-sets-dir option, 476
 - check option, 475
 - check-only-changed option, 475
 - correct-checksum option, 476
 - data-file-length option, 476
 - debug option, 474
 - description option, 478
 - example output, 350
 - extend-check option, 476, 476
 - fast option, 476
 - force option, 476, 477
 - help option, 474
 - information option, 476
 - keys-used option, 477
 - max-record-length option, 477
 - medium-check option, 476
 - no-symlinks option, 477
 - options, 474
 - parallel-recover option, 477
 - quick option, 477
 - read-only option, 476
 - recover option, 477
 - safe-recover option, 477
 - set-auto-increment[option, 478
 - set-character-set option, 477
 - set-collation option, 477
 - silent option, 474
 - sort-index option, 478
 - sort-records option, 478
 - sort-recover option, 477
 - tmpdir option, 477
 - unpack option, 478
 - update-state option, 476
 - verbose option, 474
 - version option, 474
 - wait option, 474
- mysamlog, 468
- mysampack, 468, 734, 858
 - backup option, 480
 - character-sets-dir option, 481
 - debug option, 481
 - force option, 481
 - help option, 480
 - join option, 481
 - silent option, 481
 - test option, 481
 - tmpdir option, 481
 - verbose option, 481
 - version option, 481
 - wait option, 481
- mysam_block_size mysamchk variable, 474
- mysam_ftdump, 468
 - count option, 472
 - dump option, 472
 - help option, 472
 - length option, 472
 - stats option, 472
 - verbose option, 472
- MyODBC, 1361
- MySQL
 - defined, 4
 - introduction, 4
 - pronunciation, 5
- mysql, 468
 - auto-rehash option, 485
 - batch option, 485
 - character-sets-dir option, 486
 - column-names option, 486

- column-type-info option, 486
- compress option, 486
- database option, 486
- debug option, 486
- debug-check option, 486
- debug-info option, 486
- default-character-set option, 486
- delimiter option, 486
- execute option, 486
- force option, 486
- help option, 485
- host option, 486
- html option, 486
- i-am-a-dummy option, 488
- ignore-spaces option, 486
- line-numbers option, 487
- local-infile option, 487
- named-commands option, 487
- no-auto-rehash option, 487
- no-beep option, 487
- no-named-commands option, 487
- no-pager option, 487
- no-tee option, 487
- one-database option, 487
- pager option, 487
- password option, 487
- port option, 487
- prompt option, 488
- protocol option, 488
- quick option, 488
- raw option, 488
- reconnect option, 488
- safe-updates option, 488
- secure-auth option, 488
- show-warnings option, 488
- sigint-ignore option, 488
- silent option, 488
- skip-column-names option, 488
- skip-line-numbers option, 488
- socket option, 488
- SSL options, 488
- table option, 489
- tee option, 489
- unbuffered option, 489
- user option, 489
- verbose option, 489
- version option, 489
- vertical option, 489
- wait option, 489
- xml option, 489
- MySQL AB
 - defined, 3
- MySQL binary distribution, 33
- MySQL C type, 1273
- MySQL Cluster, 987
 - administration, 1056, 1057, 1058, 1059, 1060, 1062, 1069
 - and DNS, 998
 - and IP addressing, 998
 - and networking, 999
 - and transactions, 1742, 1745
 - API node, 988, 1034
 - arbitrator, 1742, 1747
 - available platforms, 987
 - backups, 1072, 1072, 1073, 1078, 1078
 - benchmarks, 1120
 - CHECKPOINT Events, 1066
 - cluster logs, 1064, 1065
 - CLUSTERLOG commands, 1065
 - CLUSTERLOG STATISTICS command, 1069
 - commands, 1056, 1057, 1058, 1059, 1060, 1062
 - compiling from source, 1008
 - compiling with icc, 135
 - concepts, 988
 - configuration, 997, 1008, 1009, 1014, 1014, 1016, 1034, 1046, 1053, 1056
 - configuration (example), 1011
 - configuration changes, 1048
 - configuration files, 1003, 1011
 - configuration parameters, 1038, 1039, 1045, 1046
 - configuring, 1078
 - CONNECTION Events, 1066
 - connectstring, 1013
 - data node, 988, 1016
 - data types supported, 1742, 1748
 - defining node hosts, 1014
 - direct connections between nodes, 1036
 - Disk Data tables (see MySQL Cluster Disk Data)
 - ENTER SINGLE USER MODE command, 1063
 - ERROR Events, 1068
 - error logs, 1054
 - error messages, 1742, 1745
 - event log format, 1066
 - event logging thresholds, 1065
 - event logs, 1064, 1065
 - event severity levels, 1065
 - event types, 1064, 1066
 - EXIT command, 1063
 - EXIT SINGLE USER MODE command, 1063
 - FAQ, 1741
 - features new in MySQL 5.1, 1128
 - general description, 987
 - hardware requirements, 1741, 1743
 - HELP command, 1062
 - how to obtain, 1742, 1746
 - importing existing tables, 1742, 1747
 - INFO Events, 1068
 - information sources, 987
 - installation, 997, 1000, 1009
 - interconnects, 1116
 - log files, 1054
 - logging commands, 1065
 - management commands, 1069
 - management node, 988, 1014
 - managing, 1060
 - memory requirements, 1741, 1743
 - memory usage and recovery, 1048, 1122
 - mgm, 1056
 - mgm client, 1062
 - mgm management client, 1069
 - mgm process, 1060
 - mgmd, 1056
 - mgmd process, 1059
 - mysqld process, 1053, 1057
 - ndbd, 1056
 - ndbd process, 1058, 1069
 - ndb_mgm, 1004
 - ndb_size.pl (utility), 1744
 - network configuration (SCI), 1118
 - network transporters, 1116, 1117
 - networking, 1036, 1036, 1037

- networking requirements, 1741, 1742, 1743, 1745
- node failure (single user mode), 1070
- node identifiers, 1037, 1037
- node logs, 1064
- node types, 1743
- NODERESTART Events, 1067
- nodes and node groups, 989
- nodes and types, 988
- number of computers required, 1741, 1743
- obtaining, 1000
- partitioning support, 1122
- partitions, 989
- performance, 1120
- performing queries, 1005
- platforms supported, 1741, 1743
- preparing for replication, 1102
- process management, 1053
- quick configuration, 1009
- QUIT command, 1063
- replicas, 989
- replication, 1094
 - (see also MySQL Cluster replication)
- REPORT command, 1063
- requirements, 999
- resetting, 1048
- RESTART command, 1063
- restarting, 1008
- roles of computers, 1741, 1743
- runtime statistics, 1069
- SCI (Scalable Coherent Interface), 1037, 1117
- SCI drivers, 1119
- SCI network configuration, 1118
- SCI software installation, 1117
- SCI software requirements, 1117
- SCI, performance vs TCP/IP, 1120
- security, 1745
- shared memory transport, 1036
- SHOW command, 1062
- SHUTDOWN command, 1063
- shutting down, 1008
- single user mode, 1063, 1069
- SQL node, 988, 1034
- SQL nodes, 1053
- SQL statements, 1742, 1745
- SQL statements for monitoring, 1070
- START BACKUP command, 1073, 1107
- START command, 1062
- start phases (summary), 1060
- starting, 1009
- starting and stopping, 1742, 1748
- starting nodes, 1004
- starting or restarting, 1060
- STARTUP Events, 1067
- STATISTICS Events, 1068
- STATUS command, 1063
- STOP command, 1062
- storage requirements, 606
- Table is full error, 1741, 1745
- terminology, 1129
- thread states, 460
- trace files, 1055
- transaction handling, 1123
- transactions, 1018
- transporters
 - Scalable Coherent Interface (SCI), 1037
 - shared memory (SHM), 1036
 - TCP/IP, 1036
- troubleshooting backups, 1078
- upgrades and downgrades, 1048, 1048, 1050
- using in a virtual machine, 1741, 1744
- using tables and data, 1005
- vs replication, 1741, 1742
- MySQL Cluster Disk Data, 1113
 - creating log file groups, 1113
 - creating tables, 1113, 1114
 - creating tablespaces, 1114
 - dropping Disk Data objects, 1115
 - storage requirements, 1116
- MySQL Cluster Glossary, 1129
- MySQL Cluster How-To, 997
- MySQL Cluster in MySQL 5.0 and 5.1, 1128
- MySQL Cluster limitations, 1121
 - and differences from standard MySQL limits, 1122
 - autodiscovery, 1127
 - binary logging, 1125
 - character sets, 1127
 - CREATE TABLE statements, 1127
 - database objects, 1124
 - Disk Data storage, 1125
 - error handling and reporting, 1124
 - geometry data types, 1122
 - implementation, 1125
 - imposed by configuration, 1122
 - INSERT IGNORE, UPDATE IGNORE, and REPLACE statements, 1127
 - memory usage and transaction handling, 1123
 - multiple management servers, 1126
 - multiple MySQL servers, 1126
 - partitioning, 1122
 - performance, 1125
 - replication, 1122, 1127
 - resolved in current version from previous versions, 1126
 - syntax, 1121
 - transactions, 1123
 - unsupported features, 1124
- MySQL Cluster processes (types), 1053
- MySQL Cluster replication, 1094, 1105
 - and --initial option, 1100
 - and auto_increment* variables, 1100
 - and circular replication, 1096
 - and DDL, 1099
 - and multi-byte character sets, 1096
 - and primary key, 1100
 - backups, 1106, 1108
 - concepts, 1095, 1096
 - failover, 1105
 - known issues, 1096
 - loss of connection, 1096
 - preparing, 1102
 - reset-slave.pl backup automation script, 1108
 - restoring from backups, 1106
 - starting, 1103
 - storage engines other than NDB on slave, 1100
 - system tables used, 1100
- MySQL Cluster utilities, 1078
- mysql command options, 485
- mysql commands
 - list of, 490
- MySQL Dolphin name, 5
- MySQL history, 5

- mysql history file, 490
- MySQL mailing lists, 13
- MySQL name, 5
- mysql prompt command, 492
- mysql source (command for reading from text files), 163, 494
- MySQL source distribution, 33
- mysql status command, 491
- MySQL storage engines, 849
- MySQL version, 42
- mysql \. (command for reading from text files), 163, 494
- MySQL++, 1358
- mysql.event table, 1225
- mysql.ndb_binlog_index table (see MySQL Cluster replication)
- mysql.server, 182
 - basedir option, 280
 - datadir option, 280
 - pid-file option, 280
 - service-startup-timeout option, 280
 - use-manager option, 280
 - use-mysqld_safe option, 280
 - user option, 280
- mysql.sock
 - changing location of, 76
 - protection, 1785
- MYSQL323 SQL mode, 274
- MYSQL40 SQL mode, 274
- mysqlaccess, 468
 - brief option, 496
 - commit option, 496
 - copy option, 496
 - db option, 496
 - debug option, 496
 - help option, 496
 - host option, 496
 - howto option, 497
 - old_server option, 497
 - password option, 497
 - plan option, 497
 - preview option, 497
 - relnotes option, 497
 - rhost option, 497
 - rollback option, 497
 - spassword option, 497
 - superuser option, 497
 - table option, 497
 - user option, 497
 - version option, 497
- mysqladmin, 468, 716, 735, 826, 830, 833, 835
 - character-sets-dir option, 501
 - compress option, 501
 - count option, 501
 - debug option, 501
 - debug-check option, 498
 - debug-info option, 498
 - default-character-set option, 501
 - force option, 501
 - help option, 501
 - host option, 501
 - no-beep option, 501
 - password option, 501
 - port option, 501
 - protocol option, 501
 - relative option, 501
 - silent option, 501
 - sleep option, 501
 - socket option, 502
 - SSL options, 502
 - user option, 502
 - verbose option, 502
 - version option, 502
 - vertical option, 502
 - wait option, 502
- mysqladmin command options, 500
- mysqladmin option
 - mysqld_multi, 281
- mysqlbinlog, 468
 - base64-output option, 503
 - character-sets-dir option, 503
 - database option, 503
 - debug option, 503
 - debug-check option, 503
 - debug-info option, 503
 - disable-log-bin option, 503
 - force-read option, 503
 - help option, 503
 - hexdump option, 503
 - host option, 503
 - local-load option, 504
 - offset option, 504
 - password option, 504
 - port option, 504
 - position option, 504
 - protocol option, 504
 - read-from-remote-server option, 504
 - result-file option, 504
 - server-id option, 504
 - set-charset option, 504
 - short-form option, 504
 - socket option, 504
 - start-datetime option, 504
 - start-position option, 505
 - stop-datetime option, 505
 - stop-position option, 505
 - to-last-log option, 505
 - user option, 505
 - version option, 505
- mysqlbug script, 19
- mysqlcheck, 468
 - all-databases option, 509
 - all-in-1 option, 509
 - analyze option, 509
 - auto-repair option, 509
 - character-sets-dir option, 509
 - check option, 509
 - check-only-changed option, 509
 - check-upgrade option, 509
 - compress option, 509
 - databases option, 509
 - debug option, 509
 - debug-check option, 509
 - debug-info option, 509
 - default-character-set option, 509
 - extended option, 510
 - fast option, 510
 - fix-db-names option, 510
 - fix-table-names option, 510
 - force option, 510
 - help option, 508
 - host option, 510
 - medium-check option, 510

- optimize option, 510
- password option, 510
- port option, 510
- protocol option, 510
- quick option, 510
- repair option, 510
- silent option, 510
- socket option, 511
- SSL options, 511
- tables option, 511
- use-frm option, 511
- user option, 511
- verbose option, 511
- version option, 511
- write-binlog option, 505
- mysqlclient library, 1268
- mysqld, 182
 - abort-slave-event-count option, 202
 - allow-suspicious-udfs option, 202, 303
 - ansi option, 202
 - as MySQL Cluster process, 1053, 1057
 - basedir option, 202
 - bind-address option, 202
 - binlog-do-db option, 369
 - binlog-format option, 202
 - binlog-ignore-db option, 369
 - binlog-row-event-max-size option, 202
 - bootstrap option, 202
 - character-set-client-handshake option, 202
 - character-set-filesystem option, 203
 - character-set-server option, 203
 - character-sets-dir option, 202
 - chroot option, 203
 - collation-server option, 203
 - command options, 201
 - console option, 203
 - core-file option, 203
 - datadir option, 203
 - debug option, 203
 - default-character-set option, 203
 - default-collation option, 203
 - default-storage-engine option, 203
 - default-table-type option, 203
 - default-time-zone option, 203
 - delay-key-write option, 204, 856
 - des-key-file option, 204
 - disconnect-slave-event-count option, 204
 - enable-named-pipe option, 204
 - enable-pstack option, 205
 - event-scheduler option, 204
 - exit-info option, 204
 - external-locking option, 204
 - flush option, 205
 - gdb option, 205
 - general-log option, 205
 - help option, 202
 - init-file option, 205
 - innodb option, 866
 - innodb_status_file option, 866
 - language option, 205
 - large-pages option, 205
 - local-infile option, 303
 - log option, 205
 - log-bin option, 205
 - log-bin-index option, 206
 - log-bin-trust-function-creators option, 206
 - log-error option, 206
 - log-isam option, 206
 - log-long-format option, 206
 - log-output option, 206
 - log-queries-not-using-indexes option, 206
 - log-short-format option, 206
 - log-slave-updates option, 950
 - log-slow-admin-statements option, 206
 - log-slow-queries option, 206
 - log-tc option, 207
 - log-tc-size option, 207
 - log-warnings option, 207, 951
 - low-priority-updates option, 207
 - master-connect-retry option, 951
 - master-host option, 951
 - master-info-file option, 951
 - master-password option, 951
 - master-port option, 951
 - master-retry-count option, 951
 - master-ssl option, 951
 - master-ssl-ca option, 951
 - master-ssl-capath option, 951
 - master-ssl-cert option, 951
 - master-ssl-cipher option, 951
 - master-ssl-key option, 951
 - master-user option, 951
 - max-binlog-dump-events option, 207
 - max-relay-log-size option, 951
 - memlock option, 207
 - myisam-recover option, 207, 856
 - ndb-cluster-connection-pool option, 207
 - ndb-connectstring option, 208
 - ndbcluster option, 208
 - old-passwords option, 208, 304
 - one-thread option, 208
 - open-files-limit option, 208
 - pid-file option, 208
 - port option, 208
 - port-open-timeout option, 208
 - read-only option, 951
 - relay-log option, 952
 - relay-log-index option, 952
 - relay-log-info-file option, 952
 - relay-log-purge option, 952
 - relay-log-space-limit option, 952
 - replicate-do-db option, 952
 - replicate-do-table option, 952
 - replicate-ignore-db option, 953
 - replicate-ignore-table option, 953
 - replicate-rewrite-db option, 953
 - replicate-same-server-id option, 953
 - replicate-wild-do-table option, 953
 - replicate-wild-ignore-table option, 954
 - report-host option, 954
 - report-password option, 954
 - report-port option, 954
 - report-user option, 954
 - role in MySQL Cluster (see SQL Node (MySQL Cluster))
 - safe-mode option, 208
 - safe-show-database option, 209, 304
 - safe-user-create option, 209, 304
 - secure-auth option, 209, 304
 - secure-file-priv option, 209, 304
 - shared-memory option, 209

- shared-memory-base-name option, 209
- show-slave-auth-info option, 954
- skip-concurrent-insert option, 209
- skip-external-locking option, 209
- skip-grant-tables option, 209, 304
- skip-host-cache option, 209
- skip-innodb option, 210
- skip-merge option, 304
- skip-name-resolve option, 210, 304
- skip-ndbcluster option, 210
- skip-networking option, 210, 304
- skip-safemalloc option, 210
- skip-show-database option, 210, 305
- skip-slave-start option, 954
- skip-stack-trace option, 210
- skip-symbolic-links option, 210
- skip-thread-priority option, 210
- slave-load-tmpdir option, 954
- slave-net-timeout option, 955
- slave-skip-errors option, 955
- slave_compressed_protocol option, 954
- slow-query-log option, 211
- socket option, 211
- sporadic-binlog-dump-fail option, 210
- sql-mode option, 211
- SSL options, 210, 305
- standalone option, 210
- starting, 306
- symbolic-links option, 210
- sysdate-is-now option, 211
- tc-heuristic-recover option, 211
- temp-pool option, 211
- tmpdir option, 212
- transaction-isolation option, 211
- user option, 212
- version option, 212
- mysqld library, 1268
- mysqld option
 - mysqld_multi, 281
 - mysqld_safe, 277
- mysqld options, 441
- mysqld server
 - buffer sizes, 441
- mysqld-safe-compatible option
 - mysqlmanager, 286
- mysqld-version option
 - mysqld_safe, 277
- mysqldump, 104, 469
 - add-drop-database option, 512
 - add-drop-table option, 512
 - add-locks option, 512
 - all-databases option, 513
 - all-tablespaces option, 513
 - allow-keywords option, 513
 - character-sets-dir option, 513
 - comments option, 513
 - compact option, 513
 - compatible option, 513
 - complete-insert option, 513
 - compress option, 513
 - create-options option, 513
 - databases option, 513
 - debug option, 514
 - debug-check option, 514
 - debug-info option, 514
 - default-character-set option, 514
 - delayed-insert option, 514
 - delete-master-logs option, 514
 - disable-keys option, 514
 - events option, 514
 - extended-insert option, 514
 - fields-enclosed-by option, 514, 523
 - fields-escaped-by option, 514, 523
 - fields-optionally-enclosed-by option, 514, 523
 - fields-terminated-by option, 514, 523
 - first-slave option, 514
 - flush-logs option, 514
 - flush-privileges option, 514
 - force option, 515
 - help option, 512
 - hex-blob option, 515
 - host option, 515
 - ignore-table option, 515
 - insert-ignore option, 515
 - lines-terminated-by option, 515, 523
 - lock-all-tables option, 515
 - lock-tables option, 515
 - master-data option, 515
 - no-autocommit option, 515
 - no-create-db option, 516
 - no-create-info option, 516
 - no-data option, 516
 - opt option, 516
 - order-by-primary option, 516
 - password option, 516
 - port option, 516
 - problems, 520, 2069
 - protocol option, 516
 - quick option, 516
 - quote-names option, 516
 - replace option, 516
 - result-file option, 516
 - routines option, 517
 - set-charset option, 517
 - single-transaction option, 517
 - skip-comments option, 517
 - skip-opt option, 517
 - socket option, 517
 - SSL options, 517
 - tab option, 517
 - tables option, 518
 - triggers option, 518
 - tz-utc option, 518
 - user option, 518
 - verbose option, 518
 - version option, 518
 - views, 520, 2069
 - where option, 518
 - workarounds, 520, 2069
 - xml option, 518
- mysqld_multi, 182
 - config-file option, 281
 - defaults-extra-file option, 281
 - defaults-file option, 281
 - example option, 281
 - help option, 281
 - log option, 281
 - mysqladmin option, 281
 - mysqld option, 281
 - no-defaults option, 281

- no-log option, 281
- password option, 282
- silent option, 282
- tcp-ip option, 282
- user option, 282
- verbose option, 282
- version option, 282
- mysqld_safe, 182
 - autoclose option, 276
 - basedir option, 276
 - core-file-size option, 276
 - datadir option, 277
 - defaults-extra-file option, 277
 - defaults-file option, 277
 - help option, 276
 - ledir option, 277
 - log-error option, 277
 - mysqld option, 277
 - mysqld-version option, 277
 - nice option, 277
 - no-defaults option, 277
 - open-files-limit option, 277
 - pid-file option, 277
 - port option, 277
 - skip-kill-mysqld option, 277
 - skip-syslog option, 278
 - socket option, 277
 - syslog option, 278
 - syslog-tag option, 278
 - timezone option, 278
 - user option, 278
- mysqlhotcopy, 469
 - addtodest option, 521
 - allowold option, 521
 - checkpoint option, 521
 - chroot option, 521
 - debug option, 521
 - dryrun option, 521
 - flushlog option, 521
 - help option, 521
 - host option, 521
 - keepold option, 521
 - method option, 521
 - noindices option, 521
 - password option, 521
 - port option, 521
 - quiet option, 521
 - record_log_pos option, 522
 - regex option, 522
 - resetmaster option, 522
 - resetslave option, 522
 - socket option, 522
 - suffix option, 522
 - tmpdir option, 522
 - user option, 522
- mysqlimport, 104, 469, 747
 - character-sets-dir option, 523
 - columns option, 523
 - compress option, 523
 - debug option, 523
 - debug-check option, 523
 - debug-info option, 523
 - default-character-set option, 523
 - delete option, 523
 - force option, 523
 - help option, 523
 - host option, 523
 - ignore option, 523
 - ignore-lines option, 523
 - local option, 524
 - lock-tables option, 524
 - low-priority option, 524
 - password option, 524
 - port option, 524
 - protocol option, 524
 - replace option, 524
 - silent option, 524
 - socket option, 524
 - SSL options, 524
 - user option, 524
 - verbose option, 524
 - version option, 524
- mysqlmanager, 182
 - add-user option, 284
 - angel-pid-file option, 284
 - bind-address option, 284
 - check-password-file option, 284
 - clean-password-file option, 284
 - debug option, 285
 - default-mysqld-path option, 285
 - defaults-file option, 285
 - drop-user option, 285
 - edit-user option, 285
 - help option, 284
 - install option, 285
 - list-users option, 285
 - log option, 285
 - monitoring-interval option, 285
 - mysqld-safe-compatible option, 286
 - password option, 286
 - password-file option, 286
 - pid-file option, 286
 - port option, 286
 - print-defaults option, 286
 - print-password-line option, 286
 - remove option, 286
 - run-as-service option, 286
 - socket option, 286
 - standalone option, 286
 - user option, 286
 - username option, 286
 - version option, 287
 - wait-timeout option, 287
- mysqlshow, 469
 - character-sets-dir option, 525
 - compress option, 525
 - count option, 526
 - debug option, 526
 - debug-check option, 526
 - debug-info option, 526
 - default-character-set option, 526
 - help option, 525
 - host option, 526
 - keys option, 526
 - password option, 526
 - port option, 526
 - protocol option, 526
 - show-table-type option, 526
 - socket option, 526
 - SSL options, 526

- status option, 526
- user option, 526
- verbose option, 527
- version option, 527
- mysqslap, 469
 - auto-generate-sql option, 527, 528, 528
 - compress option, 527
 - concurrency option, 527
 - create option, 527
 - create-schema option, 527
 - csv option, 527
 - debug option, 527
 - debug-check option, 527
 - debug-info option, 527
 - delimiter option, 528
 - engine option, 528
 - help option, 527
 - host option, 528
 - iterations option, 528
 - lock-directory option, 528, 529
 - number-char-cols option, 528
 - number-int-cols option, 528
 - number-of-queries option, 528
 - only-print option, 528
 - password option, 528, 528
 - port option, 528
 - preserve-schema option, 528
 - protocol option, 528
 - query option, 528
 - silent option, 528
 - skip-query option, 529
 - slave option, 529
 - socket option, 529
 - SSL options, 529
 - use-threads option, 529
 - user option, 529
 - verbose option, 529
 - version option, 529
- mysqltest
 - MySQL Test Suite, 1702
- mysql_affected_rows(), 1280
- mysql_autocommit(), 1280
- MYSQL_BIND C type, 1323
- mysql_change_user(), 1281
- mysql_character_set_name(), 1282
- mysql_close(), 1282
- mysql_commit(), 1282
- mysql_connect(), 1282
- mysql_convert_table_format, 469
- mysql_create_db(), 1283
- mysql_data_seek(), 1283
- MYSQL_DEBUG environment variable, 131, 140, 470
- mysql_debug(), 1284
- mysql_drop_db(), 1284
- mysql_dump_debug_info(), 1285
- mysql_eof(), 1285
- mysql_errno(), 1286
- mysql_error(), 1287
- mysql_escape_string(), 1287
- mysql_explain_log
 - help option, 529
 - host option, 529
 - password option, 530
 - socket option, 530
 - user option, 530
- mysql_fetch_field(), 1287
- mysql_fetch_fields(), 1288
- mysql_fetch_field_direct(), 1288
- mysql_fetch_lengths(), 1289
- mysql_fetch_row(), 1289
- MYSQL_FIELD C type, 1273
- mysql_field_count(), 1290, 1302
- MYSQL_FIELD_OFFSET C type, 1273
- mysql_field_seek(), 1291
- mysql_field_tell(), 1291
- mysql_find_rows, 469
 - help option, 530
 - regexp option, 530
 - rows option, 531
 - skip-use-db option, 531
 - start_row option, 531
- mysql_fix_extensions, 469
- mysql_fix_privilege_tables, 183
- mysql_free_result(), 1291
- mysql_get_character_set_info(), 1292
- mysql_get_client_info(), 1292
- mysql_get_client_version(), 1292
- mysql_get_host_info(), 1292
- mysql_get_proto_info(), 1293
- mysql_get_server_info(), 1293
- mysql_get_server_version(), 1293
- mysql_get_ssl_cipher(), 1293
- MYSQL_GROUP_SUFFIX environment variable, 131
- mysql_hex_string(), 1294
- MYSQL_HISTFILE environment variable, 131, 490
- MYSQL_HOME environment variable, 131
- MYSQL_HOST environment variable, 131, 314
- mysql_info(), 714, 743, 754, 778, 1294
- mysql_init(), 1295
- mysql_insert_id(), 25, 744, 1295
- mysql_install_db, 183
- mysql_install_db script, 91
- mysql_kill(), 1297
- mysql_library_end(), 1297
- mysql_library_init(), 1297
- mysql_list_dbs(), 1298
- mysql_list_fields(), 1299
- mysql_list_processes(), 1299
- mysql_list_tables(), 1300
- mysql_more_results(), 1300
- mysql_next_result(), 1301
- mysql_num_fields(), 1302
- mysql_num_rows(), 1302
- mysql_options(), 1303
- mysql_ping(), 1306
- MYSQL_PS1 environment variable, 131
- MYSQL_PWD environment variable, 131, 314, 470
- mysql_query(), 1306, 1353
- mysql_real_connect(), 1307
- mysql_real_escape_string(), 539, 1310
- mysql_real_query(), 1310
- mysql_refresh(), 1311
- mysql_reload(), 1312
- MYSQL_RES C type, 1273
- mysql_rollback(), 1313
- MYSQL_ROW C type, 1273
- mysql_row_seek(), 1313
- mysql_row_tell(), 1313
- mysql_secure_installation, 183
- mysql_select_db(), 1314

- mysql_server_end(), 1352
 - mysql_server_init(), 1352
 - mysql_setpermission, 469
 - help option, 531
 - host option, 531
 - password option, 531
 - port option, 532
 - socket option, 532
 - user option, 532
 - mysql_set_character_set(), 1314
 - mysql_set_local_infile_default(), 1314, 1314
 - mysql_set_server_option(), 1316
 - mysql_shutdown(), 1317
 - mysql_sqlstate(), 1317
 - mysql_ssl_set(), 1318
 - mysql_stat(), 1318
 - MYSQL_STMT C type, 1322
 - mysql_stmt_affected_rows(), 1329
 - mysql_stmt_attr_get(), 1330
 - mysql_stmt_attr_set(), 1330
 - mysql_stmt_bind_param(), 1331
 - mysql_stmt_bind_result(), 1332
 - mysql_stmt_close(), 1332
 - mysql_stmt_data_seek(), 1333
 - mysql_stmt_erno(), 1333
 - mysql_stmt_error(), 1333
 - mysql_stmt_execute(), 1334
 - mysql_stmt_fetch(), 1336
 - mysql_stmt_fetch_column(), 1340
 - mysql_stmt_field_count(), 1341
 - mysql_stmt_free_result(), 1341
 - mysql_stmt_init(), 1341
 - mysql_stmt_insert_id(), 1341
 - mysql_stmt_num_rows(), 1342
 - mysql_stmt_param_count(), 1342
 - mysql_stmt_param_metadata(), 1342
 - mysql_stmt_prepare(), 1343
 - mysql_stmt_reset(), 1343
 - mysql_stmt_result_metadata(), 1344
 - mysql_stmt_row_seek(), 1345
 - mysql_stmt_row_tell(), 1345
 - mysql_stmt_send_long_data(), 1345
 - mysql_stmt_sqlstate(), 1347
 - mysql_stmt_store_result(), 1347
 - mysql_store_result(), 1319, 1353
 - mysql_tableinfo, 469
 - clear option, 532
 - clear-only option, 532
 - col option, 532
 - help option, 532
 - host option, 532
 - idx option, 533
 - password option, 533
 - port option, 533
 - prefix option, 533
 - quiet option, 533
 - socket option, 533
 - tbl-status option, 533
 - user option, 533
 - MYSQL_TCP_PORT environment variable, 131, 382, 382, 470
 - mysql_thread_end(), 1352
 - mysql_thread_id(), 1320
 - mysql_thread_init(), 1351
 - mysql_thread_safe(), 1352
 - MYSQL_TIME C type, 1324
 - mysql_tzinfo_to_sql, 183
 - MYSQL_UNIX_PORT environment variable, 92, 131, 382, 382, 470
 - mysql_upgrade, 183, 320
 - basedir option, 299
 - datadir option, 300
 - debug-check option, 300
 - debug-info option, 300
 - force option, 300
 - help option, 299
 - user option, 300
 - verbose option, 300
 - mysql_use_result(), 1320
 - mysql_waitpid, 469
 - help option, 533
 - verbose option, 533
 - version option, 533
 - mysql_warning_count(), 1321
 - mysql_zap, 469
 - my_bool C type, 1273
 - my_bool values
 - printing, 1273
 - my_init(), 1351
 - my_print_defaults, 468
 - config-file option, 471
 - debug option, 471
 - defaults-extra-file option, 471
 - defaults-file option, 471
 - defaults-group-suffix option, 471
 - extra-file option, 471
 - help option, 471
 - no-defaults option, 471
 - verbose option, 471
 - version option, 471
 - my_ulonglong C type, 1273
 - my_ulonglong values
 - printing, 1273
- ## N
- named pipes, 55, 59
 - named-commands option
 - mysql, 487
 - names, 540
 - case sensitivity, 542
 - variables, 550
 - NAME_CONST(), 699
 - name_file option
 - comp_err, 295
 - naming
 - releases of MySQL, 33
 - NATIONAL CHAR data type, 588
 - native functions
 - adding, 1728
 - native thread support, 32
 - NATURAL LEFT JOIN, 760
 - NATURAL LEFT OUTER JOIN, 760
 - NATURAL RIGHT JOIN, 760
 - NATURAL RIGHT OUTER JOIN, 760
 - NCHAR data type, 588
 - NDB, 1741, 1742
 - ndb option
 - perror, 535
 - NDB storage engine (see MySQL Cluster)
 - FAQ, 1741
 - ndb-cluster-connection-pool option

- mysqld, 207
- ndb-connectstring option
 - mysqld, 208
 - ndb_config, 1080
- ndbcluster option
 - mysqld, 208
- ndbd (MySQL Cluster)
 - defined, 988
 - (see also data node (MySQL Cluster))
- ndb_apply_status table (MySQL Cluster replication), 1101, 1106
 - (see also MySQL Cluster replication)
- ndb_binlog_index table (MySQL Cluster replication), 1100, 1106
 - (see also MySQL Cluster replication)
- ndb_config, 1078
 - config-file option, 1080
 - fields option, 1081
 - host option, 1081
 - id option, 1081
 - ndb-connectstring option, 1080
 - nodeid option, 1081
 - nodes option, 1081
 - query option, 1081, 1081
 - rows option, 1081
 - type option, 1081
 - usage option, 1080
 - version option, 1080
- ndb_cpcd, 1078
- ndb_delete_all, 1078
 - transactional option, 1082
- ndb_desc, 1078
 - extra-partition-info option, 1083
- ndb_drop_index, 1078
- ndb_drop_table, 1078
- ndb_error_reporter, 1078
- ndb_mgm (MySQL Cluster management node client), 1004
- ndb_mgmd (MySQL Cluster)
 - defined, 988
 - (see also management node (MySQL Cluster))
- ndb_print_backup_file, 1078
- ndb_print_schema_file, 1078
- ndb_print_sys_file, 1078
- ndb_restore
 - errors, 1077
- ndb_select_all, 1078
 - delimiter option, 1087
 - descending option, 1087
 - disk option, 1087
 - gci option, 1087
 - header option, 1087
 - lock option, 1086
 - nodata option, 1087
 - order option, 1086
 - rowid option, 1087
 - tupscan option, 1087
 - useHexFormat option, 1087
- ndb_select_count, 1078
- ndb_show_tables, 1078
 - loops option, 1088
 - parsable option, 1088
 - type option, 1089
 - unqualified option, 1089
- ndb_size.pl, 1078
- ndb_size.pl (utility), 1744
- ndb_waiter, 1078
 - no-contact option, 1093
 - not-started option, 1093
 - timeout option, 1094
- negative values, 539
- nested queries, 769
- net etiquette, 15
- netmask notation
 - in mysql.user table, 314
- NetWare, 68
- net_buffer_length variable, 490
- New features in MySQL Cluster, 1128
- new procedures
 - adding, 1729
- new users
 - adding, 71, 75
- newline (\n), 538, 751
- next-key lock, 869, 888, 890
- NFS
 - InnoDB, 861, 908
- nice option
 - mysqld_safe, 277
- no matching rows, 1789
- no-auto-rehash option
 - mysql, 487
- no-autocommit option
 - mysqldump, 515
- no-beep option
 - mysql, 487
 - mysqladmin, 501
- no-contact option
 - ndb_waiter, 1093
- no-create-db option
 - mysqldump, 516
- no-create-info option
 - mysqldump, 516
- no-data option
 - mysqldump, 516
- no-debug option
 - make_win_bin_dist, 296
- no-defaults option, 179
 - mysqld_multi, 281
 - mysqld_safe, 277
 - my_print_defaults, 471
- no-embedded option
 - make_win_bin_dist, 296
- no-log option
 - mysqld_multi, 281
- no-named-commands option
 - mysql, 487
- no-pager option
 - mysql, 487
- no-symlinks option
 - myisamchk, 477
- no-tee option
 - mysql, 487
- nodata option
 - ndb_select_all, 1087
- node groups (MySQL Cluster), 989
- node identifiers (MySQL Cluster), 1037, 1037
- node logs (MySQL Cluster), 1064
- nodeid option
 - ndb_config, 1081
- NODERESTART Events (MySQL Cluster), 1067
- nodes option
 - ndb_config, 1081
- noindices option

- mysqlhotcopy, 521
- non-delimited strings, 595
- Non-transactional tables, 1789
- NoOfDiskPagesToDiskAfterRestartACC
 - calculating, 1046
- NoOfDiskPagesToDiskAfterRestartACC (DEPRECATED), 1029
- NoOfDiskPagesToDiskAfterRestartTUP
 - calculating, 1046
- NoOfDiskPagesToDiskAfterRestartTUP (DEPRECATED), 1028
- NoOfDiskPagesToDiskDuringRestartACC (DEPRECATED), 1029
- NoOfDiskPagesToDiskDuringRestartTUP (DEPRECATED), 1029
- NoOfFragmentLogFiles, 1022
 - calculating, 1046
- NoOfReplicas, 1017
- NOT
 - logical, 624
- NOT BETWEEN, 622
- not equal (!=), 621
- not equal (<>), 621
- NOT EXISTS
 - with subqueries, 772
- NOT IN, 623
- NOT LIKE, 638
- NOT NULL
 - constraint, 28
- NOT REGEXP, 639
- not-started option
 - ndb_waiter, 1093
- Novell NetWare, 68
- NOW(), 662
- NO_AUTO_CREATE_USER SQL mode, 271
- NO_AUTO_VALUE_ON_ZERO SQL mode, 271
- NO_BACKSLASH_ESCAPES SQL mode, 271
- NO_DIR_IN_CREATE SQL mode, 271
- NO_FIELD_OPTIONS SQL mode, 272
- NO_KEY_OPTIONS SQL mode, 272
- NO_TABLE_OPTIONS SQL mode, 272
- NO_UNSIGNED_SUBTRACTION SQL mode, 272
- NO_ZERO_DATE SQL mode, 272
- NO_ZERO_IN_DATE SQL mode, 273
- NUL, 537, 751
- NULL, 156, 1787
 - ORDER BY, 413, 757
 - testing for null, 620, 621, 622, 626
- NULL value, 156, 540
- NULL values
 - and AUTO_INCREMENT columns, 1788
 - and indexes, 725
 - and TIMESTAMP columns, 1788
 - vs. empty values, 1787
- NULLIF(), 627
- number-char-cols option
 - mysqslap, 528
- number-int-cols option
 - mysqslap, 528
- number-of-queries option
 - mysqslap, 528
- numbers, 539
- NUMERIC data type, 587
- numeric types, 606
- numeric-dump-file option
 - resolve_stack_dump, 536
- NumGeometries(), 1184
- NumInteriorRings(), 1183
- NumPoints(), 1181

O

- obtaining information about partitions, 1155
- Obtaining MySQL Cluster, 1000
- OCT(), 633
- OCTET_LENGTH(), 633
- ODBC, 1361
- ODBC compatibility, 542, 586, 619, 621, 723, 762
- ODirect, 1025
- offset option
 - mysqlbinlog, 504
- OLAP, 703
- old-passwords option
 - mysqld, 208, 304
- OLD_PASSWORD(), 691
- old_server option
 - mysqlaccess, 497
- ON DUPLICATE KEY, 741
- one-database option
 - mysql, 487
- one-thread option
 - mysqld, 208
- online location of manual, 1
- only-debug option
 - make_win_bin_dist, 296
- only-print option
 - mysqslap, 528
- ONLY_FULL_GROUP_BY
 - SQL mode, 705
- ONLY_FULL_GROUP_BY SQL mode, 273
- OPEN, 1201
- Open Source
 - defined, 4
- open tables, 439, 500
- open-files-limit option
 - mysqld, 208
 - mysqld_safe, 277
- OpenGIS, 1166
- opening
 - tables, 439
- Opening master dump table
 - thread state, 460
- Opening mysql.ndb_apply_status
 - thread state, 460
- Opening table
 - thread state, 454
- Opening tables
 - thread state, 454
- opens, 500
- OpenSSL, 334, 334
- open_files_limit variable, 505
- operating systems
 - file-size limits, 1776
 - supported, 32
 - Windows versus Unix, 61
- operations
 - arithmetic, 645
- operators, 611
 - assignment, 550
 - cast, 644, 678
 - logical, 624
 - precedence, 618
- opt option
 - mysqldump, 516

- optimization
 - subquery, 415
 - tips, 422
- optimizations, 397, 401
- optimize option
 - mysqlcheck, 510
- OPTIMIZE TABLE, 803
- optimizer
 - controlling, 445
- optimizing
 - DISTINCT, 415
 - filesort, 412
 - GROUP BY, 413
 - LEFT JOIN, 404
 - LIMIT, 419
 - tables, 350
- option files, 176, 320
- options
 - command-line
 - mysql, 485
 - mysqladmin, 500
 - configure, 75
 - embedded server, 1269
 - libmysqld, 1269
 - myisamchk, 474
 - provided by MySQL, 144
 - replication, 967
- OR, 167, 401
 - bitwise, 686
 - logical, 625
- OR Index Merge optimization, 401
- Oracle compatibility, 21, 703, 779
- ORACLE SQL mode, 274
- ORD(), 633
- ORDER BY, 153, 711, 757
 - NULL, 413, 757
- order option
 - ndb_select_all, 1086
- order-by-primary option
 - mysqldump, 516
- OUTFILE, 758
- out_dir option
 - comp_err, 295
- out_file option
 - comp_err, 295
- Overlaps(), 1187
- overview, 1

P

- packages
 - list of, 2081
- pager option
 - mysql, 487
- parallel-recover option
 - myisamchk, 477
- parameters
 - server, 441
- parentheses (and), 618
- parsable option
 - ndb_show_tables, 1088
- PARTITION, 1132
- partition management, 1148
- partition pruning, 1157
- partitioning, 1132
 - advantages, 1134
 - and dates, 1135
 - and foreign keys, 1160
 - and FULLTEXT indexes, 1160
 - and key cache, 1160
 - and server SQL mode, 1160
 - and subqueries, 1160
 - and temporary tables, 1160, 1160
 - by hash, 1139
 - by key, 1141
 - by linear hash, 1140
 - by linear key, 1142
 - by list, 1138
 - by range, 1136
 - concepts, 1132
 - data type of partitioning key, 1160
 - enabling, 1133
 - functions disallowed in partitioning expressions, 1164
 - functions supported in partitioning expressions, 1163
 - limitations, 1159
 - operators disallowed in partitioning expressions, 1160
 - operators supported in partitioning expressions, 1160
 - optimization, 1156, 1157
 - resources, 1132
 - storage engines (limitations), 1163
 - subpartitioning, 1160
 - support, 1133
 - support in MySQL Cluster, 1122
 - types, 1135
- Partitioning
 - maximum number of partitions, 1160
- partitioning information statements, 1155
- partitioning keys and primary keys, 1160
- partitioning keys and unique keys, 1160
- partitions
 - adding and dropping, 1148
 - analyzing, 1154
 - checking, 1154
 - managing, 1148
 - modifying, 1148
 - optimizing, 1154
 - repairing, 1154
 - splitting and merging, 1148
- PARTITIONS
 - INFORMATION_SCHEMA table, 1246
- partitions (MySQL Cluster), 989
- password
 - root user, 96
- password encryption
 - reversibility of, 691
- password option
 - mysql, 487
 - mysqlaccess, 497
 - mysqladmin, 501
 - mysqlbinlog, 504
 - mysqlcheck, 510
 - mysqldump, 516
 - mysqld_multi, 282
 - mysqlhotcopy, 521
 - mysqlimport, 524
 - mysqlmanager, 286
 - mysqlshow, 526
 - mysqslap, 528, 528
 - mysql_explain_log, 530
 - mysql_setpermission, 531

- mysql_tableinfo, 533
- PASSWORD(), 315, 332, 691, 1778
- password-file option
 - mysqlmanager, 286
- passwords
 - for users, 327
 - forgotten, 1780
 - lost, 1780
 - resetting, 1780
 - security, 306
 - setting, 332, 796, 799
- PATH environment variable, 71, 131, 173
- pattern matching, 156, 639
- performance
 - benchmarks, 386
 - disk issues, 464
 - estimating, 396
 - improving, 429, 976, 977
- PERIOD_ADD(), 662
- PERIOD_DIFF(), 662
- Perl
 - installing, 131
 - installing on Windows, 132
- Perl API, 1358
- Perl DBI/DBD
 - installation problems, 133
- permission checks
 - effect on speed, 387
- perror, 469
 - ndb option, 1745
 - help option, 534
 - ndb option, 535
 - silent option, 535
 - verbose option, 535
 - version option, 535
- PHP API, 1357
- PI(), 650
- pid-file option
 - mysql.server, 280
 - mysqld, 208
 - mysqld_safe, 277
 - mysqlmanager, 286
- Ping
 - thread command, 452
- PIPES_AS_CONCAT SQL mode, 273
- plan option
 - mysqlaccess, 497
- plugin API, 1703
- PLUGINS
 - INFORMATION_SCHEMA table, 1246
- plugins
 - adding, 1703
 - installing, 1705
 - uninstalling, 1706
- POINT data type, 1174
- point in time recovery, 345
- Point(), 1176
- PointFromText(), 1175
- PointFromWKB(), 1176
- PointN(), 1182
- PointOnSurface(), 1184
- PolyFromText(), 1175
- PolyFromWKB(), 1176
- POLYGON data type, 1174
- Polygon(), 1177
- PolygonFromText(), 1175
- PolygonFromWKB(), 1176
- port option
 - mysql, 487
 - mysqladmin, 501
 - mysqlbinlog, 504
 - mysqlcheck, 510
 - mysqld, 208
 - mysqldump, 516
 - mysqld_safe, 277
 - mysqlhotcopy, 521
 - mysqlimport, 524
 - mysqlmanager, 286
 - mysqlshow, 526
 - mysqslap, 528
 - mysql_setpermission, 532
 - mysql_tableinfo, 533
- port-open-timeout option
 - mysqld, 208
- portability, 384
 - types, 609
- porting
 - to other systems, 135
- PortNumber, 1015, 1036
- position option
 - mysqlbinlog, 504
- POSITION(), 633
- post-install
 - multiple servers, 377
- post-installation
 - setup and testing, 87
- PostgreSQL compatibility, 22
- POSTGRESQL SQL mode, 274
- POW(), 650
- POWER(), 650
- precedence
 - operator, 618
- precision
 - arithmetic, 1261
- precision math, 1261
- prefix option
 - mysql_tableinfo, 533
- Prepare
 - thread command, 452
- PREPARE, 846, 846
 - XA transactions, 789
- preparing
 - thread state, 455
- preserve-schema option
 - mysqslap, 528
- preview option
 - mysqlaccess, 497
- previx option
 - configure, 75
- PRIMARY KEY, 711, 724
 - constraint, 28
- primary key
 - deleting, 711
- primary keys
 - and partitioning keys, 1160
- print-defaults option, 179
 - mysqlmanager, 286
- print-password-line option
 - mysqlmanager, 286
- privilege

- changes, 319
- privilege information
 - location, 310
- privilege system, 306
 - described, 306
- privileges
 - access, 306
 - adding, 328
 - default, 96
 - deleting, 331, 791
 - display, 820
 - dropping, 331, 791
 - granting, 792
 - revoking, 799
- problems
 - access denied errors, 1768
 - common errors, 1767
 - compiling, 81
 - DATE columns, 1786
 - date values, 595
 - installing on IBM-AIX, 119
 - installing on Solaris, 111
 - installing Perl, 133
 - linking, 1779
 - ODBC, 1439
 - reporting, 16
 - starting the server, 94
 - table locking, 427
 - time zone, 1786
- PROCEDURE, 759
- procedures
 - adding, 1729
 - stored, 25, 1191
- process management (MySQL Cluster), 1053
- process support, 32
- processes
 - display, 824
- processing
 - arguments, 1723
- Processing events
 - thread state, 460
- Processing events from schema table
 - thread state, 460
- Processlist
 - thread command, 452
- PROCESLIST, 824
 - INFORMATION_SCHEMA table, 1256
- program variables
 - setting, 180
- programs
 - client, 468, 1355
 - crash-me, 384
 - server side, 182
 - utility, 468
- prompt option
 - mysql, 488
- prompts
 - meanings, 146
- pronunciation
 - MySQL, 5
- protocol option
 - mysql, 488
 - mysqladmin, 501
 - mysqlbinlog, 504
 - mysqlcheck, 510

- mysqldump, 516
- mysqlimport, 524
- mysqlshow, 526
- mysqslap, 528
- PURGE MASTER LOGS, 837
- PURGE STALE SESSIONS, 1061
- Purging old relay logs
 - thread state, 455
- Python API, 1359

Q

- QUARTER(), 662
- queries
 - entering, 145
 - estimating performance, 396
 - examples, 163
 - speed of, 387
 - Twin Studies project, 169
- Query
 - thread command, 452
- Query Cache, 445
- query end
 - thread state, 455
- query option
 - mysqslap, 528
 - ndb_config, 1081, 1081
- questions, 500
 - answering, 15
- Queueing master event to the relay log
 - thread state, 459
- quick option
 - myisamchk, 477
 - mysql, 488
 - mysqlcheck, 510
 - mysqldump, 516
- quiet option
 - mysqlhotcopy, 521
 - mysql_tableinfo, 533
- Quit
 - thread command, 452
- QUIT command (MySQL Cluster), 1063
- QUOTE(), 634
- quote-names option
 - mysqldump, 516
- quotes
 - in strings, 538
- quoting, 539
- quoting binary data, 538
- quoting of identifiers, 541

R

- RADIANS(), 650
- RAND(), 650
- range partitioning, 1136
- range partitions
 - adding and dropping, 1149
 - managing, 1149
- raw option
 - mysql, 488
- re-creating
 - grant tables, 92
- read-from-remote-server option
 - mysqlbinlog, 504
- read-only option

- mysamchk, 476
- mysqld, 951
- Reading event from the relay log
 - thread state, 459
- Reading from net
 - thread state, 455
- Reading master dump table data
 - thread state, 460
- read_buffer_size myisamchk variable, 474
- REAL data type, 586
- REAL_AS_FLOAT SQL mode, 273
- Rebuilding the index on master dump table
 - thread state, 460
- ReceiveBufferMemory, 1036
- reconfiguring, 81, 81
- reconnect option
 - mysql, 488
- Reconnecting after a failed binlog dump request
 - thread state, 459
- Reconnecting after a failed master event read
 - thread state, 459
- record_log_pos option
 - mysqlhotcopy, 522
- RECOVER
 - XA transactions, 789
- recover option
 - mysamchk, 477
- recovery
 - from crash, 347
 - point in time, 345
- RedoBuffer, 1030
- reducing
 - data size, 429
- references, 711
- REFERENTIAL_CONSTRAINTS
 - INFORMATION_SCHEMA table, 1257
- Refresh
 - thread command, 452
- ref_or_null, 403
- REGEXP, 639
- REGEXP operator, 639
- regexp option
 - mysqlhotcopy, 522
 - mysql_find_rows, 530
- Register Slave
 - thread command, 452
- Registering slave on master
 - thread state, 458
- regular expression syntax, 639
- Related(), 1187
- relational databases
 - defined, 4
- relative option
 - mysqladmin, 501
- relay-log option
 - mysqld, 952
- relay-log-index option
 - mysqld, 952
- relay-log-info-file option
 - mysqld, 952
- relay-log-purge option
 - mysqld, 952
- relay-log-space-limit option
 - mysqld, 952
- release numbers, 33
- RELEASE SAVEPOINT, 784
- releases
 - naming scheme, 33
 - testing, 34
 - updating, 35
- RELEASE_LOCK(), 699
- relnotes option
 - mysqlaccess, 497
- remove option
 - mysqlmanager, 286
- Removing duplicates
 - thread state, 455
- removing tmp table
 - thread state, 455
- rename
 - thread state, 455
- RENAME DATABASE, 737
- rename result table
 - thread state, 455
- RENAME TABLE, 737
- RENAME USER, 798
- renaming user accounts, 798
- Reopen tables
 - thread state, 455
- reordering
 - columns, 1792
- Repair by sorting
 - thread state, 455
- Repair done
 - thread state, 455
- repair option
 - mysqlcheck, 510
- repair options
 - mysamchk, 476
- REPAIR TABLE, 804
- Repair with keycache
 - thread state, 455
- repairing
 - tables, 348
- REPEAT, 1203
- REPEAT(), 634
- replace, 470
- REPLACE, 754
- replace option
 - mysqldump, 516
 - mysqlimport, 524
- REPLACE(), 634
- replicas (MySQL Cluster), 989
- replicate-do-db option
 - mysqld, 952
- replicate-do-table option
 - mysqld, 952
- replicate-ignore-db option
 - mysqld, 953
- replicate-ignore-table option
 - mysqld, 953
- replicate-rewrite-db option
 - mysqld, 953
- replicate-same-server-id option
 - mysqld, 953
- replicate-wild-do-table option
 - mysqld, 953
- replicate-wild-ignore-table option
 - mysqld, 954
- replication, 938

- circular, 1096
- in MySQL Cluster, 1094
 - (see also MySQL Cluster replication)
- replication implementation, 980
- replication limitations, 967
- replication master
 - thread states, 458
- replication masters
 - statements, 837
- replication options, 967
- replication slave
 - thread states, 458, 459, 460
- replication slaves
 - statements, 839
- replication, asynchronous (see MySQL Cluster replication)
- REPORT command (MySQL Cluster), 1063
- report-host option
 - mysqld, 954
- report-password option
 - mysqld, 954
- report-port option
 - mysqld, 954
- report-user option
 - mysqld, 954
- reporting
 - bugs, 16
 - Connector/NET problems, 1605
 - Connector/ODBC problems, 1439, 1439
 - errors, 1, 16
- Requesting binlog dump
 - thread state, 458
- REQUIRE GRANT option, 797
- reschedule
 - thread state, 458
- reserved words, 547
- RESET MASTER, 837
- RESET SLAVE, 841
- Reset stmt
 - thread command, 452
- reset-slave.pl (see MySQL Cluster replication)
- resetmaster option
 - mysqlhotcopy, 522
- resetslave option
 - mysqlhotcopy, 522
- resolveip, 470
 - help option, 536
 - silent option, 536
 - version option, 536
- resolve_stack_dump, 470
 - help option, 536
 - numeric-dump-file option, 536
 - symbols-file option, 536
 - version option, 536
- RESTART command (MySQL Cluster), 1063
- restarting
 - the server, 90
- RestartOnErrorInsert, 1025
- RESTORE TABLE, 805
- restoring from backups
 - in MySQL Cluster replication, 1106
- restrictions
 - server-side cursors, 2066
 - stored routines, 2064
 - subqueries, 2066
 - triggers, 2064
 - views, 2068
- result-file option
 - mysqlbinlog, 504
 - mysqldump, 516
- retrieving
 - data from tables, 150
- return (\r), 538, 751
- return values
 - UDFs, 1725
- REVERSE(), 634
- REVOKE, 799
- revoking
 - privileges, 799
- rhost option
 - mysqlaccess, 497
- RIGHT JOIN, 760
- RIGHT OUTER JOIN, 760
- RIGHT(), 634
- RLIKE, 639
- ROLLBACK, 23, 782
 - XA transactions, 789
- rollback option
 - mysqlaccess, 497
- ROLLBACK TO SAVEPOINT, 784
- Rolling back
 - thread state, 455
- rolling restart (MySQL Cluster), 1048
- rolling upgrades and downgrades (MySQL Cluster), 1048
- ROLLUP, 703
- root password, 96
- root user
 - password resetting, 1780
- ROUND(), 651
- rounding, 1261
- rounding errors, 585
- ROUTINES
 - INFORMATION_SCHEMA table, 1242
- routines option
 - mysqldump, 517
- ROW, 772
- row subqueries, 772
- row-level locking, 425
- rowid option
 - ndb_select_all, 1087
- rows
 - counting, 158
 - deleting, 1789
 - locking, 25
 - matching problems, 1789
 - selecting, 151
 - sorting, 153
- rows option
 - mysql_find_rows, 531
 - ndb_config, 1081
- ROW_COUNT(), 696
- RPAD(), 634
- RPM file, 63
- RPM Package Manager, 63
- RTRIM(), 634
- RTS-threads, 141
- run-as-service option
 - mysqlmanager, 286
- running
 - ANSI mode, 20
 - batch mode, 162

- multiple servers, 377
- queries, 145
- running configure after prior invocation, 81

S

- safe-mode option
 - mysqld, 208
- safe-recover option
 - myisamchk, 477
- safe-show-database option
 - mysqld, 209, 304
- safe-updates option, 495
 - mysql, 488
- safe-user-create option
 - mysqld, 209, 304
- Sakila, 5
- SAVEPOINT, 784
- Saving state
 - thread state, 455
- scale
 - arithmetic, 1261
- schema
 - altering, 707
 - creating, 716
 - deleting, 735
- SCHEMA(), 697
- SCHEMATA
 - INFORMATION_SCHEMA table, 1235
- SCHEMA_PRIVILEGES
 - INFORMATION_SCHEMA table, 1239
- SCI (Scalable Coherent Interface) (see MySQL Cluster)
- script files, 162
- scripts
 - mysqlbug, 19
 - mysql_install_db, 91
- searching
 - and case sensitivity, 1786
 - full-text, 668
 - MySQL Web pages, 16
 - two keys, 167
- Searching rows for update
 - thread state, 455
- SECOND(), 663
- secure-auth option
 - mysql, 488
 - mysqld, 209, 304
- secure-file-priv option
 - mysqld, 209, 304
- security
 - against attackers, 302
- security system, 306
- SEC_TO_TIME(), 663
- SELECT
 - LIMIT, 755
 - optimizing, 387
 - Query Cache, 445
- SELECT INTO, 1198
- SELECT INTO TABLE, 23
- SELECT speed, 396
- selecting
 - databases, 148
- select_limit variable, 490
- SendBufferMemory, 1035
- Sending binlog event to slave
 - thread state, 458
- SendLimit, 1038
- SendSignalId, 1035, 1037, 1038
- SEQUENCE, 167
- sequence emulation, 696
- sequences, 167
- SERIAL, 585
- SERIAL DEFAULT VALUE, 599
- server
 - connecting, 144, 313
 - debugging, 136
 - disconnecting, 144
 - restart, 90
 - shutdown, 90
 - starting, 88
 - starting and stopping, 93
 - starting problems, 94
- server variables, 212, 830
- server-id option
 - mysqlbinlog, 504
- server-side cursor restrictions, 2066
- server-side programs, 182
- ServerPort, 1016
- servers
 - multiple, 377
- service-startup-timeout option
 - mysql.server, 280
- SESSION_STATUS
 - INFORMATION_SCHEMA table, 1258
- SESSION_USER(), 697
- SESSION_VARIABLES
 - INFORMATION_SCHEMA table, 1258
- SET, 805, 1198
 - AUTOCOMMIT, 805
 - BIG_TABLES, 805
 - CHARACTER SET, 561, 805
 - FOREIGN_KEY_CHECKS, 805
 - IDENTITY, 805
 - INSERT_ID, 805
 - LAST_INSERT_ID, 805
 - NAMES, 561, 805
 - ONE_SHOT, 805
 - size, 609
 - SQL_AUTO_IS_NULL, 805
 - SQL_BIG_SELECTS, 805
 - SQL_BUFFER_SELECT, 805
 - SQL_LOG_BIN, 805
 - SQL_LOG_OFF, 805
 - SQL_LOG_UPDATE, 805
 - SQL_NOTES, 805
 - SQL_QUOTE_SHOW_CREATE, 805
 - SQL_SAFE_UPDATES, 805
 - SQL_SELECT_LIMIT, 805
 - SQL_WARNINGS, 805
 - TIMESTAMP, 805
 - UNIQUE_CHECKS, 805
- SET data type, 590, 604
- SET GLOBAL SQL_SLAVE_SKIP_COUNTER, 842
- Set option
 - thread command, 452
- SET OPTION, 805
- SET PASSWORD, 799
- SET PASSWORD statement, 332
- SET SQL_LOG_BIN, 837
- SET TRANSACTION, 787

- set-auto-increment[option
 - myisamchk, 478
- set-character-set option
 - myisamchk, 477
- set-charset option
 - mysqlbinlog, 504
 - mysqldump, 517
- set-collation option
 - myisamchk, 477
- setting
 - passwords, 332
- setting passwords, 799
- setting program variables, 180
- setup
 - post-installation, 87
 - thread state, 456
- SHA(), 691
- SHA1(), 691
- shared memory transporter (see MySQL Cluster)
- shared-memory option
 - mysqld, 209
- shared-memory-base-name option
 - mysqld, 209
- SharedBufferSize, 1038
- SharedGlobalMemory, 1116
- shell syntax, 3
- ShmKey, 1037
- ShmSize, 1037
- short-form option
 - mysqlbinlog, 504
- SHOW
 - in MySQL Cluster management client, 1011
- SHOW AUTHORS, 810, 811
- SHOW BINARY LOGS, 838
- SHOW BINLOG EVENTS, 810, 837
- SHOW CHARACTER SET, 810, 811
- SHOW COLLATION, 810, 811
- SHOW COLUMNS, 810, 812
- SHOW command (MySQL Cluster), 1062
- SHOW CONTRIBUTORS, 810, 812
- SHOW CREATE DATABASE, 810, 812
- SHOW CREATE EVENT, 810
- SHOW CREATE FUNCTION, 810, 813
- SHOW CREATE PROCEDURE, 810, 813
- SHOW CREATE SCHEMA, 810, 812
- SHOW CREATE TABLE, 810, 813
- SHOW CREATE TRIGGER, 810, 813
- SHOW CREATE VIEW, 810, 814
- SHOW DATABASES, 810, 814
- SHOW ENGINE, 810, 814
 - used with MySQL Cluster, 1070
- SHOW ENGINE INNODB STATUS, 814
- SHOW ENGINE NDB STATUS, 814, 1070
- SHOW ENGINE NDBCLUSTER STATUS, 814, 1070
- SHOW ENGINES, 810, 817
 - used with MySQL Cluster, 1070
- SHOW ERRORS, 810, 818
 - and MySQL Cluster, 1745
- SHOW EVENTS, 810
- SHOW extensions, 1258
- SHOW FIELDS, 810
- SHOW FUNCTION CODE, 810, 823
- SHOW FUNCTION STATUS, 810, 823
- SHOW GRANTS, 810, 820
- SHOW INDEX, 810, 821
- SHOW INNODB STATUS, 810
- SHOW KEYS, 810, 821
- SHOW MASTER LOGS, 810, 838
- SHOW MASTER STATUS, 810, 838
- SHOW OPEN TABLES, 810, 822
- SHOW PLUGINS, 810, 822
- SHOW PRIVILEGES, 810, 823
- SHOW PROCEDURE CODE, 810, 823
- SHOW PROCEDURE STATUS, 810, 823
- SHOW PROCESSLIST, 810, 824
- SHOW SCHEDULER STATUS, 810, 1222
- SHOW SCHEMAS, 810, 814
- SHOW SLAVE HOSTS, 810, 838
- SHOW SLAVE STATUS, 810, 842
- SHOW STATUS, 810
 - used with MySQL Cluster, 1072
- SHOW STORAGE ENGINES, 817
- SHOW TABLE STATUS, 810
- SHOW TABLE TYPES, 810, 817
- SHOW TABLES, 810, 829
- SHOW TRIGGERS, 810, 829
- SHOW VARIABLES, 810
 - used with MySQL Cluster, 1071
- SHOW WARNINGS, 810, 831
 - and MySQL Cluster, 1745
- SHOW with WHERE, 1234, 1258
- show-slave-auth-info option
 - mysqld, 954
- show-table-type option
 - mysqlshow, 526
- show-warnings option
 - mysql, 488
- Shutdown
 - thread command, 452
- SHUTDOWN command (MySQL Cluster), 1063
- shutdown_timeout variable, 502
- shutting down
 - the server, 90
- Shutting down
 - thread state, 460
- sigint-ignore option
 - mysql, 488
- SIGN(), 652
- silent column changes, 733
- silent option
 - myisamchk, 474
 - myisampack, 481
 - mysql, 488
 - mysqladmin, 501
 - mysqlcheck, 510
 - mysqld_multi, 282
 - mysqlimport, 524
 - mysqslap, 528
 - perlor, 535
 - resolveip, 536
- SIN(), 652
- single quote (\'), 537
- single user mode (MySQL Cluster), 1063, 1069
 - and ndb_restore, 1074
- single-transaction option
 - mysqldump, 517
- size of tables, 1776
- sizes
 - display, 584
- skip-column-names option

- mysql, 488
- skip-comments option
 - mysqldump, 517
- skip-concurrent-insert option
 - mysqld, 209
- skip-external-locking option
 - mysqld, 209
- skip-grant-tables option
 - mysqld, 209, 304
- skip-host-cache option
 - mysqld, 209
- skip-innodb option
 - mysqld, 210
- skip-kill-mysqld option
 - mysqld_safe, 277
- skip-line-numbers option
 - mysql, 488
- skip-merge option
 - mysqld, 304
- skip-name-resolve option
 - mysqld, 210, 304
- skip-ndbcluster option
 - mysqld, 210
- skip-networking option
 - mysqld, 210, 304
- skip-opt option
 - mysqldump, 517
- skip-query option
 - mysqslap, 529
- skip-safemalloc option
 - mysqld, 210
- skip-show-database option
 - mysqld, 210, 305
- skip-slave-start option
 - mysqld, 954
- skip-stack-trace option
 - mysqld, 210
- skip-symbolic-links option
 - mysqld, 210
- skip-syslog option
 - mysqld_safe, 278
- skip-thread-priority option
 - mysqld, 210
- skip-use-db option
 - mysql_find_rows, 531
- slave option
 - mysqslap, 529
- slave-load-tmpdir option
 - mysqld, 954
- slave-net-timeout option
 - mysqld, 955
- slave-skip-errors option
 - mysqld, 955
- slave_allow_batching, 1104
- slave_compressed_protocol option
 - mysqld, 954
- Sleep
 - thread command, 452
- sleep option
 - mysqladmin, 501
- SLEEP(), 699
- slow queries, 500
- slow query log, 375
- slow-query-log option
 - mysqld, 211
- SMALLINT data type, 585
- socket location
 - changing, 76
- socket option
 - mysql, 488
 - mysqladmin, 502
 - mysqlbinlog, 504
 - mysqlcheck, 511
 - mysqld, 211
 - mysqldump, 517
 - mysqld_safe, 277
 - mysqlhotcopy, 522
 - mysqlexport, 524
 - mysqlmanager, 286
 - mysqshow, 526
 - mysqslap, 529
 - mysql_explain_log, 530
 - mysql_setpermission, 532
 - mysql_tableinfo, 533
- Solaris
 - installation, 67
- Solaris installation problems, 111
- Solaris troubleshooting, 82
- Solaris x86_64 issues, 894
- SOME, 770
- sort-index option
 - myisamchk, 478
- sort-records option
 - myisamchk, 478
- sort-recover option
 - myisamchk, 477
- sorting
 - character sets, 355
 - data, 153
 - grant tables, 316, 317
 - table rows, 153
- Sorting for group
 - thread state, 456
- Sorting for order
 - thread state, 456
- Sorting index
 - thread state, 456
- Sorting result
 - thread state, 456
- sort_buffer_size myisamchk variable, 474
- sort_key_blocks myisamchk variable, 474
- SOUNDEX(), 635
- SOUNDS LIKE, 635
- source (mysql client command), 163, 494
- source distribution
 - installing, 72
- source distributions
 - on Linux, 106
- SPACE(), 635
- spassword option
 - mysqlaccess, 497
- Spatial Extensions in MySQL, 1166
- speed
 - compiling, 461
 - increasing with replication, 938
 - inserting, 420
 - linking, 461
 - of queries, 387, 396
- sporadic-binlog-dump-fail option
 - mysqld, 210

- SQL
 - defined, 4
- SQL mode
 - ONLY_FULL_GROUP_BY, 705
- SQL node (MySQL Cluster)
 - defined, 988
- SQL nodes (MySQL Cluster), 1053
- SQL statements
 - replication masters, 837
 - replication slaves, 839
- SQL statements relating to MySQL Cluster, 1070
- SQL-92
 - extensions to, 19
- sql-mode option
 - mysqld, 211
- SQL_BIG_RESULT, 760
- SQL_BUFFER_RESULT, 760
- SQL_CACHE, 447, 760
- SQL_CALC_FOUND_ROWS, 760
- SQL_NO_CACHE, 447, 760
- SQL_SMALL_RESULT, 760
- sql_yacc.cc problems, 81
- SQRT(), 652
- square brackets, 584
- SRID(), 1180
- SSH, 340
- SSL, 334
- SSL and X509 Basics, 334
- SSL command options, 336
- ssl option, 336
- SSL options
 - mysql, 488
 - mysqldadmin, 502
 - mysqlcheck, 511
 - mysqld, 210, 305
 - mysqldump, 517
 - mysqldimport, 524
 - mysqlshow, 526
 - mysqlslap, 529
- SSL related options, 797
- ssl-ca option, 337
- ssl-capath option, 337
- ssl-cert option, 337
- ssl-cipher option, 337
- ssl-key option, 337
- ssl-verify-server-cert option, 337
- standalone option
 - mysqld, 210
 - mysqlmanager, 286
- Standard SQL
 - differences from, 23, 798
 - extensions to, 19, 20
- standards compatibility, 19
- START
 - XA transactions, 789
- START BACKUP command (MySQL Cluster), 1073
- START command (MySQL Cluster), 1062
- START SLAVE, 845
- START TRANSACTION, 782
- start-datetime option
 - mysqlbinlog, 504
- start-position option
 - mysqlbinlog, 505
- StartFailureTimeout, 1026
- starting
 - comments, 26
 - mysqld, 306
 - the server, 88
 - the server automatically, 93
- Starting many servers, 377
- starting slave
 - thread state, 460
- StartPartialTimeout, 1026
- StartPartitionedTimeout, 1026
- StartPoint(), 1182
- STARTUP Events (MySQL Cluster), 1067
- startup options
 - default, 176
- startup parameters, 441
 - mysql, 485
 - mysqldadmin, 500
 - tuning, 440
- start_row option
 - mysql_find_rows, 531
- statefile option
 - comp_err, 295
- statements
 - GRANT, 328
 - INSERT, 329
 - replication masters, 837
 - replication slaves, 839
- statically
 - compiling, 76
- Statistics
 - thread command, 452
- statistics
 - thread state, 456
- STATISTICS
 - INFORMATION_SCHEMA table, 1238
- STATISTICS Events (MySQL Cluster), 1068
- stats option
 - myisam_ftdump, 472
- stats_method myisamchk variable, 474
- status
 - tables, 827
- status command
 - results, 500
- STATUS command (MySQL Cluster), 1063
- status option
 - mysqlshow, 526
- status variables, 254, 826
- STD(), 703
- STDDEV(), 703
- STDDEV_POP(), 703
- STDDEV_SAMP(), 703
- STOP command (MySQL Cluster), 1062
- STOP SLAVE, 846
- stop-datetime option
 - mysqlbinlog, 505
- stop-position option
 - mysqlbinlog, 505
- StopOnError, 1025
- stopping
 - the server, 93
- stopword list
 - user-defined, 677
- storage engine
 - ARCHIVE, 922
- storage engines
 - choosing, 849

- storage of data, 429
- storage requirements
 - data type, 606
- storage space
 - minimizing, 429
- stored procedures, 1191
- stored procedures and triggers
 - defined, 25
- stored routine restrictions, 2064
- stored routines
 - LAST_INSERT_ID(), 1204
- storing row into queue
 - thread state, 457
- STRAIGHT_JOIN, 760, 760
- STRCMP(), 639
- STRICT SQL mode, 270
- STRICT_ALL_TABLES SQL mode, 273
- STRICT_TRANS_TABLES SQL mode, 270, 273
- string collating, 358
- string comparison functions, 637
- string comparisons
 - case sensitivity, 637
- string functions, 627
- string literal introducer, 537, 558
- string types, 600
- StringMemory, 1019
- strings
 - defined, 537
 - escaping characters, 537
 - non-delimited, 595
- stripping
 - defined, 464
- STR_TO_DATE(), 663
- SUBDATE(), 663
- subpartitioning, 1143
- subpartitions, 1143
- subqueries, 769
 - correlated, 773
 - errors, 774
 - rewriting as joins, 776
 - with ALL, 771
 - with ANY, IN, SOME, 770
 - with EXISTS, 772
 - with NOT EXISTS, 772
 - with ROW, 772
- subquery, 769
- subquery optimization, 415
- subquery restrictions, 2066
- subselects, 769
- SUBSTR(), 635
- SUBSTRING(), 635
- SUBSTRING_INDEX(), 636
- SUBTIME(), 663
- subtraction (-), 645
- suffix option
 - mysqlhotcopy, 522
- SUM(), 703
- SUM(DISTINCT), 703
- superuser, 96
- superuser option
 - mysqlaccess, 497
- support
 - for operating systems, 32
- suppression
 - default values, 28

- Sybase compatibility, 782
- symbolic links, 465, 467
- symbolic-links option
 - mysqld, 210
- symbols-file option
 - resolve_stack_dump, 536
- SymDifference(), 1185
- Syncing ndb table schema operation and binlog
 - thread state, 460
- syntax
 - regular expression, 639
- SYSDATE(), 664
- sysdate-is-now option
 - mysqld, 211
- syslog option
 - mysqld_safe, 278
- syslog-tag option
 - mysqld_safe, 278
- system
 - privilege, 306
 - security, 300
- System lock
 - thread state, 456
- system optimization, 440
- system table, 389
- system variables, 212, 246, 830
- SYSTEM_USER(), 697

T

- tab (t), 538, 751
- tab option
 - mysqldump, 517
- table
 - changing, 708, 711, 1791
 - deleting, 736
- table aliases, 756
- table cache, 439
- Table Dump
 - thread command, 453
- table is full, 807, 1776
- Table is full error
 - MySQL Cluster, 1741, 1745
- Table lock
 - thread state, 456
- table names
 - case sensitivity, 542
 - case-sensitivity, 21
- table option
 - mysql, 489
 - mysqlaccess, 497
- table scans
 - avoiding, 420
- table types
 - choosing, 849
- table-level locking, 425
- tables
 - BLACKHOLE, 924
 - changing column order, 1792
 - checking, 475
 - closing, 439
 - compressed format, 858
 - constant, 389, 397
 - copying, 732
 - counting rows, 158

- creating, 148
- CSV, 923
- defragment, 858
- defragmenting, 355, 803
- deleting rows, 1789
- displaying status, 827
- dynamic, 858
- error checking, 347
- EXAMPLE, 916
- FEDERATED, 917
- flush, 500
- fragmentation, 803
- grant, 319
- HEAP, 915
- host, 318
- improving performance, 429
- information, 350
- information about, 161
- InnoDB, 860
- loading data, 150
- maintenance schedule, 355
- maximum size, 1776
- MEMORY, 915
- MERGE, 911
- merging, 911
- multiple, 160
- MyISAM, 854
- names, 540
- open, 439
- opening, 439
- optimizing, 350
- partitioning, 911
- repairing, 348
- retrieving data, 150
- selecting columns, 152
- selecting rows, 151
- sorting rows, 153
- symbolic links, 466
- system, 389
- too many, 440
- unique ID for last row, 1354
- updating, 23
- TABLES
 - INFORMATION_SCHEMA table, 1236
- tables option
 - mysqlcheck, 511
 - mysqldump, 518
- table_open_cache, 439
- TABLE_PRIVILEGES
 - INFORMATION_SCHEMA table, 1239
- TAN(), 652
- tar
 - problems on Solaris, 67, 111
- tbl-status option
 - mysql_tableinfo, 533
- tc-heuristic-recover option
 - mysqld, 211
- Tcl API, 1359
- tcp-ip option
 - mysqld_multi, 282
- TCP/IP, 55, 59
- tee option
 - mysql, 489
- temp-pool option
 - mysqld, 211
- temporary file
 - write access, 92
- temporary tables
 - internal, 463
 - problems, 1792
- terminal monitor
 - defined, 144
- test option
 - myisampack, 481
- testing
 - connection to the server, 314
 - installation, 88
 - of MySQL releases, 34
 - post-installation, 87
- testing mysqld
 - mysqltest, 1702
- TEXT
 - size, 608
- TEXT columns
 - default values, 602
 - indexing, 430, 725
- TEXT data type, 589, 602
- text files
 - importing, 494
- thread command
 - Binlog Dump, 451
 - Change user, 451
 - Close stmt, 451
 - Connect, 451
 - Connect Out, 451
 - Create DB, 451
 - Daemon, 451
 - Debug, 451
 - Delayed insert, 451
 - Drop DB, 451
 - Error, 451
 - Execute, 451
 - Fetch, 451
 - Field List, 452
 - Init DB, 452
 - Kill, 452
 - Long Data, 452
 - Ping, 452
 - Prepare, 452
 - Processlist, 452
 - Query, 452
 - Quit, 452
 - Refresh, 452
 - Register Slave, 452
 - Reset stmt, 452
 - Set option, 452
 - Shutdown, 452
 - Sleep, 452
 - Statistics, 452
 - Table Dump, 453
 - Time, 453
- thread commands, 451
- thread packages
 - differences between, 142
- thread state
 - After create, 453
 - allocating local table, 457
 - Analyzing, 453
 - Changing master, 460
 - Checking master version, 458

- Checking table, 453
- cleaning up, 453
- Clearing, 461
- closing tables, 453
- Committing events to binlog, 460
- Connecting to master, 458
- converting HEAP to MyISAM, 453
- copy to tmp table, 453
- Copying to group table, 453
- Copying to tmp table, 453
- Copying to tmp table on disk, 453
- Creating delayed handler, 457
- Creating index, 453
- Creating sort index, 453
- creating table, 454
- Creating table from master dump, 460
- Creating tmp table, 454
- deleting from main table, 454
- deleting from reference tables, 454
- discard_or_import_tablespace, 454
- end, 454
- Execution of init_command, 454
- Finished reading one binlog; switching to next binlog, 458
- Flushing tables, 454
- freeing items, 454
- FULLTEXT initialization, 454
- got handler lock, 457
- got old table, 457
- Has read all relay log; waiting for the slave I/O thread to update it, 459
- Has sent all binlog to slave; waiting for binlog to be updated, 458
- init, 454
- Initialized, 461
- insert, 457
- Killed, 454
- Killing slave, 460
- Locked, 454
- logging slow query, 454
- login, 454
- Making temp file, 459
- Opening master dump table, 460
- Opening mysql.ndb_apply_status, 460
- Opening table, 454
- Opening tables, 454
- preparing, 455
- Processing events, 460
- Processing events from schema table, 460
- Purging old relay logs, 455
- query end, 455
- Queueing master event to the relay log, 459
- Reading event from the relay log, 459
- Reading from net, 455
- Reading master dump table data, 460
- Rebuilding the index on master dump table, 460
- Reconnecting after a failed binlog dump request, 459
- Reconnecting after a failed master event read, 459
- Registering slave on master, 458
- Removing duplicates, 455
- removing tmp table, 455
- rename, 455
- rename result table, 455
- Reopen tables, 455
- Repair by sorting, 455
- Repair done, 455
- Repair with keycache, 455
- Requesting binlog dump, 458
- reschedule, 458
- Rolling back, 455
- Saving state, 455
- Searching rows for update, 455
- Sending binlog event to slave, 458
- setup, 456
- Shutting down, 460
- Sorting for group, 456
- Sorting for order, 456
- Sorting index, 456
- Sorting result, 456
- starting slave, 460
- statistics, 456
- storing row into queue, 457
- Syncing ndb table schema operation and binlog, 460
- System lock, 456
- Table lock, 456
- update, 457
- Updating, 456
- updating main table, 456
- updating reference tables, 456
- upgrading lock, 458
- User lock, 456
- waiting for delay_list, 457
- Waiting for event from ndbcluster, 460
- Waiting for first event from ndbcluster, 461
- waiting for handler insert, 457
- waiting for handler lock, 457
- waiting for handler open, 457
- Waiting for INSERT, 458
- Waiting for master to send event, 459
- Waiting for master update, 458
- Waiting for ndbcluster binlog update to reach current position, 461
- Waiting for ndbcluster to start, 461
- Waiting for next activation, 461
- Waiting for scheduler to stop, 461
- Waiting for schema epoch, 461
- Waiting for slave mutex on exit, 459, 460
- Waiting for table, 456
- Waiting for tables, 456
- Waiting for the next event in relay log, 459
- Waiting for the slave SQL thread to free enough relay log space, 459
- Waiting on cond, 456
- Waiting on empty queue, 461
- Waiting to finalize termination, 458
- Waiting to reconnect after a failed binlog dump request, 459
- Waiting to reconnect after a failed master event read, 459
- Writing to net, 457
- thread states
 - delayed inserts, 457
 - event scheduler, 461
 - general, 453
 - MySQL Cluster, 460
 - replication master, 458
 - replication slave, 458, 459, 460
- thread support, 32
 - non-native, 83
- threaded clients, 1355
- threads, 500, 824, 1702
 - display, 824
 - RTS, 141
- Time
 - thread command, 453
- TIME data type, 587, 598

- time types, 607
 - time zone problems, 1786
 - TIME(), 664
 - TimeBetweenGlobalCheckpoints, 1027, 1027
 - TimeBetweenInactiveTransactionAbortCheck (MySQL Cluster configuration parameter), 1028
 - TimeBetweenLocalCheckpoints, 1027
 - TimeBetweenWatchDogCheck, 1026
 - TimeBetweenWatchDogCheckInitial, 1026
 - TIMEDIFF(), 664
 - timeout, 226, 698, 746
 - connect_timeout variable, 490, 502
 - shutdown_timeout variable, 502
 - timeout option
 - ndb_waiter, 1094
 - TIMESTAMP
 - and NULL values, 1788
 - TIMESTAMP data type, 587, 594
 - TIMESTAMP(), 664
 - TIMESTAMPADD(), 664
 - TIMESTAMPDIFF(), 665
 - timezone option
 - mysqld_safe, 278
 - TIME_FORMAT(), 665
 - TIME_TO_SEC(), 665
 - TINYBLOB data type, 589
 - TINYINT data type, 584
 - TINYTEXT data type, 589
 - tips
 - optimization, 422
 - TMPDIR environment variable, 92, 131
 - tmpdir option
 - myisamchk, 477
 - myisampack, 481
 - mysqld, 212
 - mysqlhotcopy, 522
 - to-last-log option
 - mysqlbinlog, 505
 - TODO
 - symlinks, 466
 - tools
 - list of, 2082
 - Touches(), 1187
 - TO_DAYS(), 665
 - trace DBI method, 138
 - trace files (MySQL Cluster), 1055
 - TRADITIONAL SQL mode, 270, 274
 - transaction-isolation option
 - mysqld, 211
 - transaction-safe tables, 23, 860
 - transactional option
 - ndb_delete_all, 1082
 - TransactionBufferMemory, 1021
 - TransactionDeadlockDetectionTimeout (MySQL Cluster configuration parameter), 1028
 - TransactionInactiveTimeout (MySQL Cluster configuration parameter), 1028
 - transactions
 - support, 23, 860
 - Translators
 - list of, 2079
 - trigger restrictions, 2064
 - trigger, creating, 1208
 - trigger, dropping, 1211
 - triggers, 25, 829, 1208
 - LAST_INSERT_ID(), 1204
 - TRIGGERS
 - INFORMATION_SCHEMA table, 1244
 - triggers option
 - mysqldump, 518
 - TRIM(), 636
 - troubleshooting
 - FreeBSD, 82
 - Solaris, 82
 - TRUE, 539, 540
 - testing for, 621
 - TRUNCATE, 777
 - and MySQL Cluster, 1122
 - TRUNCATE(), 652
 - tupscan option
 - ndb_select_all, 1087
 - tutorial, 144
 - Twin Studies
 - queries, 169
 - type conversions, 618, 620
 - type option
 - ndb_config, 1081
 - ndb_show_tables, 1089
 - types
 - column, 584
 - columns, 609
 - data, 584
 - date, 607
 - Date and Time, 593
 - numeric, 606
 - of tables, 849
 - portability, 609
 - strings, 600
 - time, 607
 - typographical conventions, 2
 - TZ environment variable, 131, 1786
 - tz-utc option
 - mysqldump, 518
- ## U
- UCASE(), 636
 - UCS-2, 554
 - UDFs, 1719, 1719
 - compiling, 1726
 - defined, 1718
 - return values, 1725
 - ulimit, 1779
 - UMASK environment variable, 131, 1780
 - UMASK_DIR environment variable, 131, 1780
 - unary minus (-), 645
 - unbuffered option
 - mysql, 489
 - UNCOMPRESS(), 691
 - UNCOMPRESSED_LENGTH(), 691
 - UndoDataBuffer, 1030
 - UndoIndexBuffer, 1030
 - UNHEX(), 636
 - Unicode, 554
 - Unicode Collation Algorithm, 575
 - UNINSTALL PLUGIN, 1706
 - uninstalling plugins, 1706
 - UNION, 167, 767
 - Union(), 1185
 - UNIQUE, 711

- constraint, 28
- unique ID, 1354
- unique keys
 - and partitioning keys, 1160
- Unix, 1361, 1440
- UNIX_TIMESTAMP(), 665
- UNKNOWN
 - testing for, 621
- unloading
 - tables, 150
- UNLOCK TABLES, 785
- unnamed views, 773
- unpack option
 - myisamchk, 478
- unqualified option
 - ndb_show_tables, 1089
- UNTIL, 1203
- update
 - thread state, 457
- UPDATE, 777
- update-state option
 - myisamchk, 476
- UpdateXML(), 682
- updating
 - releases of MySQL, 35
 - tables, 23
- Updating
 - thread state, 456
- updating main table
 - thread state, 456
- updating reference tables
 - thread state, 456
- upgrades
 - MySQL Cluster, 1048, 1048, 1050
- upgrades and downgrades (MySQL Cluster)
 - compatibility between versions, 1050
- upgrading, 99
 - different architecture, 103
 - to 5.1, 100
- upgrading lock
 - thread state, 458
- UPPER(), 637
- uptime, 500
- URLs for downloading MySQL, 42
- usage option
 - ndb_config, 1080
- USE, 781
- USE INDEX, 766
- USE KEY, 766
- use-frm option
 - mysqlcheck, 511
- use-manager option
 - mysql.server, 280
- use-mysqld_safe option
 - mysql.server, 280
- use-threads option
 - mysqslap, 529
- useHexFormat option
 - ndb_select_all, 1087
- user accounts
 - creating, 791
 - renaming, 798
- USER environment variable, 131, 314
- User lock
 - thread state, 456

- user option
 - mysql, 489
 - mysql.server, 280
 - mysqlaccess, 497
 - mysqladmin, 502
 - mysqlbinlog, 505
 - mysqlcheck, 511
 - mysqld, 212
 - mysqldump, 518
 - mysqld_multi, 282
 - mysqld_safe, 278
 - mysqlhotcopy, 522
 - mysqlimport, 524
 - mysqlmanager, 286
 - mysqlshow, 526
 - mysqslap, 529
 - mysql_explain_log, 530
 - mysql_setpermission, 532
 - mysql_tableinfo, 533
 - mysql_upgrade, 300
- user privileges
 - adding, 328
 - deleting, 331, 791
 - dropping, 331, 791
- user table
 - sorting, 316
- user variables, 550
- USER(), 697
- user-defined functions
 - adding, 1718, 1720
- User-defined functions, 1719, 1719
- username option
 - mysqlmanager, 286
- usernames
 - and passwords, 327
- users
 - adding, 71, 75
 - deleting, 331, 791
 - root, 96
- USER_PRIVILEGES
 - INFORMATION_SCHEMA table, 1238
- uses
 - of MySQL, 385
- using multiple disks to start data, 467
- UTC_DATE(), 666
- UTC_TIME(), 666
- UTC_TIMESTAMP(), 666
- UTF-8, 554
- utility programs, 468
- UUID(), 699

V

- valid numbers
 - examples, 539
- VALUES(), 700
- VARBINARY data type, 589, 601
- VARCHAR
 - size, 608
- VARCHAR data type, 589, 600
- VARCHARACTER data type, 589
- variables
 - mysqld, 441
 - server, 212, 830
 - status, 254, 826

- system, 212, 246, 830
- user, 550
- VARIANCE(), 703
- VAR_POP(), 703
- VAR_SAMP(), 703
- verbose option
 - myisamchk, 474
 - myisampack, 481
 - mysam_ftdump, 472
 - mysql, 489
 - mysqladmin, 502
 - mysqlcheck, 511
 - mysqldump, 518
 - mysqld_multi, 282
 - mysqlexport, 524
 - mysqlshow, 527
 - mysqslap, 529
 - mysql_upgrade, 300
 - mysql_waitpid, 533
 - my_print_defaults, 471
 - perror, 535
- version
 - choosing, 33
 - latest, 42
- version option
 - comp_err, 296
 - myisamchk, 474
 - myisampack, 481
 - mysql, 489
 - mysqlaccess, 497
 - mysqladmin, 502
 - mysqlbinlog, 505
 - mysqlcheck, 511
 - mysqld, 212
 - mysqldump, 518
 - mysqld_multi, 282
 - mysqlexport, 524
 - mysqlmanager, 287
 - mysqlshow, 527
 - mysqslap, 529
 - mysql_waitpid, 533
 - my_print_defaults, 471
 - ndb_config, 1080
 - perror, 535
 - resolveip, 536
 - resolve_stack_dump, 536
- VERSION(), 697
- vertical option
 - mysql, 489
 - mysqladmin, 502
- Vietnamese, 1749, 1758
- view restrictions, 2068
- views, 26, 1227, 1227
 - updatable, 26, 1227
- VIEWS
 - INFORMATION_SCHEMA table, 1243
- Views
 - limitations, 2069
 - privileges, 2069
 - problems, 2069
- virtual memory
 - problems while compiling, 81
- Vision, 1435
- Visual Objects, 1433
- Visual Studio, 85

- Visual Studio Plugin, 1361

W

- wait option
 - myisamchk, 474
 - myisampack, 481
 - mysql, 489
 - mysqladmin, 502
- wait-timeout option
 - mysqlmanager, 287
- waiting for delay_list
 - thread state, 457
- Waiting for event from ndbcluster
 - thread state, 460
- Waiting for first event from ndbcluster
 - thread state, 461
- waiting for handler insert
 - thread state, 457
- waiting for handler lock
 - thread state, 457
- waiting for handler open
 - thread state, 457
- Waiting for INSERT
 - thread state, 458
- Waiting for master to send event
 - thread state, 459
- Waiting for master update
 - thread state, 458
- Waiting for ndbcluster binlog update to reach current position
 - thread state, 461
- Waiting for ndbcluster to start
 - thread state, 461
- Waiting for next activation
 - thread state, 461
- Waiting for scheduler to stop
 - thread state, 461
- Waiting for schema epoch
 - thread state, 461
- Waiting for slave mutex on exit
 - thread state, 459, 460
- Waiting for table
 - thread state, 456
- Waiting for tables
 - thread state, 456
- Waiting for the next event in relay log
 - thread state, 459
- Waiting for the slave SQL thread to free enough relay log space
 - thread state, 459
- Waiting on cond
 - thread state, 456
- Waiting on empty queue
 - thread state, 461
- Waiting to finalize termination
 - thread state, 458
- Waiting to reconnect after a failed binlog dump request
 - thread state, 459
- Waiting to reconnect after a failed master event read
 - thread state, 459
- WEEK(), 666
- WEEKDAY(), 667
- WEEKOFYEAR(), 667
- Well-Known Binary format, 1173
- Well-Known Text format, 1172
- WHERE, 397

- with SHOW, 1234, 1258
- where option
 - mysqldump, 518
- WHILE, 1203
- widths
 - display, 584
- Wildcard character (%), 538
- Wildcard character (_), 538
- wildcards
 - and LIKE, 433
 - in mysql.columns_priv table, 317
 - in mysql.db table, 317
 - in mysql.host table, 317
 - in mysql.procs_priv table, 317
 - in mysql.tables_priv table, 317
 - in mysql.user table, 314
- Windows, 1361, 1440
 - compiling on, 87
 - open issues, 63
 - upgrading, 60
 - versus Unix, 61
- with-big-tables option, 75
 - configure, 78
- with-debug option
 - configure, 77
- with-embedded-server option
 - configure, 75
- with-extra-charsets option
 - configure, 77
- with-unix-socket-path option
 - configure, 76
- Within(), 1187
- without-server option, 75
 - configure, 75
- WKB format, 1173
- WKT format, 1172
- wrappers
 - Eiffel, 1359
- write access
 - tmp, 92
- write-binlog option
 - mysqlcheck, 505
- write_buffer_size myisamchk variable, 474
- Writing to net
 - thread state, 457

X

- X(), 1181
- X509/Certificate, 334
- XA BEGIN, 789
- XA COMMIT, 789
- XA PREPARE, 789
- XA RECOVER, 789
- XA ROLLBACK, 789
- XA START, 789
- XA transactions, 788
 - transaction identifiers, 789
- xid
 - XA transaction identifier, 789
- xml option
 - mysql, 489
 - mysqldump, 518
- XOR
 - bitwise, 687

- logical, 625

Y

- Y(), 1181
- yaSSL, 334, 334
- Year 2000 compliance, 599
- Year 2000 issues, 599
- YEAR data type, 587, 599
- YEAR(), 667
- YEARWEEK(), 668
- Yen sign (Japanese), 1749, 1752